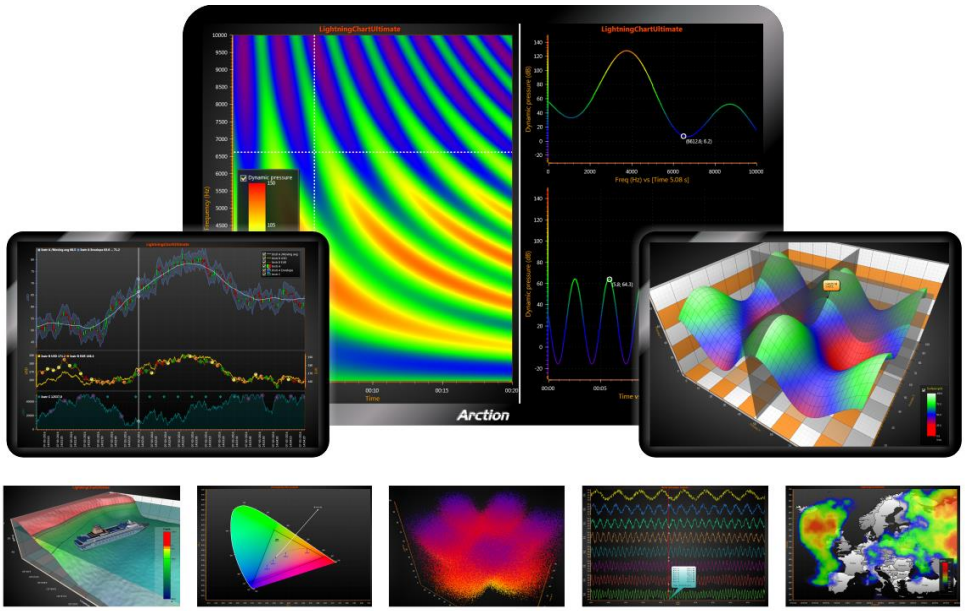


LightningChart®

簡略版取扱説明書

本資料について

この資料は、***ArctionLightningChart®.NET*** の取扱説明書の簡易版です。重要な主要機能のみが説明されています。本資料では、数百のクラス、プロパティ、メソッドについて説明していません。特定の機能についての情報や詳細については、より広範囲な内容をカバーしている英語版の取扱説明書を参照してください。また、インターアクティブエグザンプルデモアプリケーションを実行して、多くの LightningChart の機能のクイックレビューを入手してください。デモアプリケーションのソースコードは、コードで LightningChart コンポーネントを使用する方法を理解するのに役立ちます。

他の言語での取扱説明書は、下記 LightningChart. NET Resources ウェブページを参照ください。
(https://www.arction.com/lightningchart-net-resources/).

あるいは、ご質問の際は、お気軽にサポート(support@arction.com)までご連絡ください。

LightningChart .NET, v.10.1 に対応

www.arction.com

www.lightningchart.com

目次

1. 概要

LightningChart®.NET SDK は Microsoft Visual Studio のアドオンであり、WPF (*Windows Presentation Foundation*) 、UWP (*Universal Windows Platform)* および *Windows Forms* .NET プラットフォーム用のデータ視覚化関連ソフトウェアコンポーネントとツールクラスで構成されています。

Arction コンポーネントは、科学、エンジニアリング、測定や取引のソリューション、実行パフォーマンス、特殊なフォーカスにおける高度な機能などに提供されています。

LightningChart コンポーネントは、低速の GDI / GDI +または WPF Graphics API の代わりに低レベルの DirectX11 および DirectX9 GPU アクセラレーションを使用します。LightningChart は、一部の仮想マシンプラットフォームなどで GPU にアクセスできない場合、DirectX11 / DirectX10 WARP ソフトウェアレンダリングにフォールバックします。

1.1 チャートエディション

WPF の場合、異なるパフォーマンスと MVVM（モデル-ビュー-ビューモデル）のバインド可能性のバランスをとるために、LightningChart コンポーネントは、様々なバインディングレベルエディションで利用できます。

チャートエディション	プロパティのバインディング	シリーズデータのバインディング	データポイントごとのバインディング	パフォーマンス
WPF (バインド不可)	不可	不可	不可	素晴らしい
WPF (セミバインド可能)	可	可	不可	とても良い
WPF (バインド可能)	可	可	可	良い
WinForms	不可	不可	不可	最良

- WPF での最高パフォーマンスとマルチスレッドの活用には、バインド不可のチャートを選択します。
- WPF のバインド可能性とパフォーマンスをバランス良く両立するには、バインド可能のチャートを選択します。バインド可能は、MVVM のデザインパターンもサポートします。

セミバインド可能なチャート API は LightningChart v.6 の WPF チャートに非常に似ていますが、それにコレクションで作成されたオブジェクトもカバーする拡張プロパティバインディングが付属しています。

同じアプリケーションで異なるチャートエディションを使用できます。パフォーマンスが重要なタスクにバインド不可のチャートを使用しながら、バインド可能なチャートで基本的なチャートを作成し、バインドする事が出来ます。セミバインド可能およびバインド可能なチャート (ViewXY 軸、3D ライトなど）のコレクションプロパティはデフォルトで空であり、XAML エディ

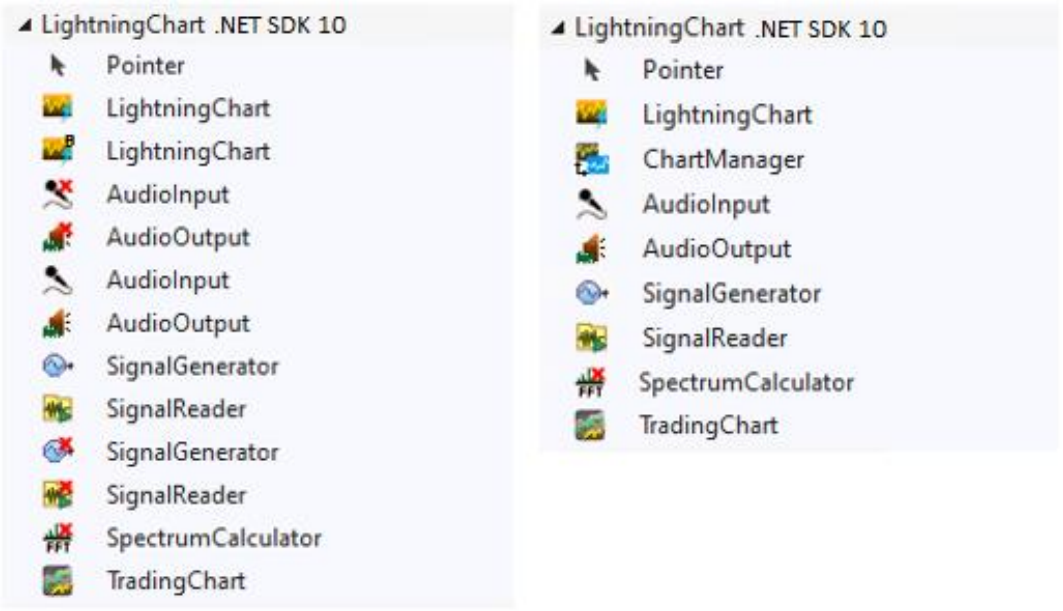
ターを完全にサポートしています。バインド不可や WinForms のコレクションでは、デフォルトのアイテムが事前に入力されています。

プリデータポイントのバインディングは、完成にバインド可能な WPF のみでサポートされます。これは、ソースコードで入手出来ます。**ソースコード**を持っているクライアントのみしか、そこで作成する事は出来ません。8.5 が、Arction によってサポートされていて、データ毎にポイントバインディング機能を持っている物の最新版です。

注意：バインド不可の WPF チャートは、XAML での構成に適応していません。コードビハインドで使用してください。

1.2 コンポーネント

UI を持たないコンポーネントには、**X** のマークが付いています。



左は、WPF ツールボックスコンポーネント、右は、 WinForms ツールボックスコンポーネント

チャート作成 アセンブリ

LightningChart チャートコンポーネント。 様々なプレゼンテーションでデータを視覚化します。

*アイコン上隅の**B = バインド可能 WPF チャート***

UWP チャート UWP アプリケーションで使用できます。

ChartManager 複数のチャートコンポーネントの相互運用とリアルタイム測メリ管理を制御します。第 17 章を参照してください。

TradingCharts アセンブリ

TradingChart トレーディングおよびファイナンスアプリ用に作られたチャート作成管理です。トレーダーライブラリが、LightningChart API に加えて作成されます。第 18 章を参照してください。

SignalTools アセンブリ

UI を持たないコンポーネントには、**X** のマークが付いています。

AudioInput サウンドデバイスから波形オーディオストリームを読み取ります。ライン入力またはマイク入力コネクターは、サウンドデバイスで使用できる一般的なオプションです。リアルタイムストリームは、他のコントロールに転送できます。第 0 章を参照してください。

AudioOutput サウンドデバイスを介して、たとえばスピーカーやライン出力などのリアルタイムデータストリームを再生します。オーディオストリームである必要はなく、サンプリングされたリアルタイム信号を使用できます。第 0 章を参照してください。

SignalGenerator 複数の構成可能な波形コンポーネントから信号を生成します。第 0 章を参照してください。

SignalReader PCM 形式の WAV などの信号ファイルから波形データを読み取ります。第 0 章を参照してください。

SpectrumCalculator FFT（高速フーリエ変換）を使用して、信号データ（時間領域）をスペクトル（周波数領域）に変換します。周波数ドメインから時間ドメインへの逆方向変換のメソッドも含まれています。第 0 章を参照してください。

1.3 名前空間

チャートエディション	アセンブリ名	名前空間のルート	XML 名前空間
WPF (バインド不可)	Arction.**Wpf.Charting**.LightningChart.dll	Arction.Wpf.Charting	xmlns:lcunb="http://schemas.arction.com/**charting**/ultimate/"
WPF (バインド可能)	Arction.**Wpf.ChartingMVVM**.LightningChart.dll	Arction.Wpf.ChartingMVVM	xmlns:lcusb="http://schemas.arction.com/ChartingMVVM/ultimate/"
UWP	Arction.**Uwp.ChartingMVVM**.LightningChart.dll	Arction.Uwp.ChartingMVVM	xmlns:lcu="using:Arction.Uwp.ChartingMVVM"
WinForms	Arction.**WinForms.Charting**.LightningChart.dll	Arction.WinForms.Charting	N/A

UWP は XML において、数個の名前空間を使用します。下記が典型的なものです。

```
xmlns:lcu="using:Arction.Uwp.ChartingMVVM"
xmlns:viewxy="using:Arction.Uwp.ChartingMVVM.Views.ViewXY"
xmlns:axes="using:Arction.Uwp.ChartingMVVM.Axes"
xmlns:titles="using:Arction.Uwp.ChartingMVVM.Titles"
xmlns:seriesxy="using:Arction.Uwp.ChartingMVVM.SeriesXY"
```

ViewXY 以外のビューを使用する際は、それぞれのビューとシリーズ名を使用します (View3D, ViewPolar 等) 。

2. インストール

コンピューターの構成が要件を満たしているかどうかを確認します。

- グラフィックハードウェアなしでレンダリングするための DirectX 9.0c（シェーダーモデル 3）レベルのグラフィックアダプター以上、または DirectX11 互換のオペレーティングシステム。DirectX11 互換のグラフィックスハードウェアが推奨されています。
- 32 ビットまたは 64 ビットの Windows Vista 7、8 または 10、Windows Server 2008 R2 以降の物
- 開発用の Visual Studio 2010-2017、展開には不要
- .NET framework v. 4.0 以降のインストール

LightningChart .NET SDK v10.exe を右クリックします。セットアップにより、コンポーネントが Visual Studio ツールボックスにインストールされます。また、ツールボックスコントロールに

関連付けられたヘルプファイルもインストールされます。コンポーネントやヘルプのインストールに失敗した場合は、次のセクションの指示に従って手動でインストールしてください。LightningChart を試用する場合、**SetupDownloader.exe** が最も多く使用されます。これにより SDK がダウンロード、インストールされ、LightningChart .NET SDK v10.exe を実行する必要はありません。

2.1 Arction components を手動で Visual Studio Toolbox に追加する

WinForms

1. Visual Studio を開きます。新しい **WinForms** プロジェクトを作成します。[ツールボックス]を右クリックし、[**タブの追加**] を選択して「Arction」という名前を付けます。

2. [Arction]タブを右クリックして、 [**アイテムの選択...**] を選択します。
3. [**ツールボックスアイテムの選択**] ウィンドウで、 [**.NET Framework コンポーネント**] ページを選択します。 [**参照**] をクリックします。

   コンポーネントがインストールされたフォルダ（通常は ***C:\Program Files (x86)\Arction\LightningChart .NET SDK v.10\LibNet4***）から **Arction.WinForms.Charting.LightningChartUltimate.dll** および **Arction.WinForms.SignalProcessing.SignalTools.dll** を参照し、クリックして開きます。これでコンポーネントがツールボックスに表示されます。

WPF

1. Visual Studio を開きます。新しい **WPF** プロジェクトを作成します。[ツールボックス]を右クリックし、[**タブの追加**] を選択して「Arction」という名前を付けます。
2. [Arction]タブを右クリックして、 [**アイテムの選択...**] を選択します。
3. [**ツールボックスアイテムの選択**] ウィンドウで、[**WPF コンポーネント**] ページを選択します。 [**参照...**] をクリックします。

   コンポーネントがインストールされたフォルダ（通常は ***C:\Program Files (x86)\Arction\LightningChart .NET SDK v.10\LibNet4***）から **Arction.Wpf.Charting.LightningChartUltimate.dll**、**Arction.Wpf.SemibindableCharting.LightningChartUltimate.dll**、**Arction.Wpf.BindableCharting.LightningChartUltimate.dll**、**Arction.Wpf.SignalProcessing.SignalTools.dll** を参照し、クリックして開きます。これでコンポーネントがツールボックスに表示されます。

2.2 Visual Studio 2010-2019 のヘルプを手動で構成する

この章では、LightningChart®.NET ヘルプコンテンツを手動でインストールする方法について説明します。Visual Studio 2010-2019 にローカルヘルプコンテンツがインストールされていない場合、この情報が必要になります。LightningChart®.NET をインストールする際にローカルヘルプコンテンツがインストールされていない場合、LightningChart®.NET のヘルプはインストールされません。

以下の手順により、Visual Studio 2010-2019 から LightningChart®.NET のヘルプを表示できます。LightningChar のクラス、プロパティなどで F1 キーを押すか、Microsoft ヘルプビューアーを使用してヘルプコンテンツを参照します。

Visual Studio 2010

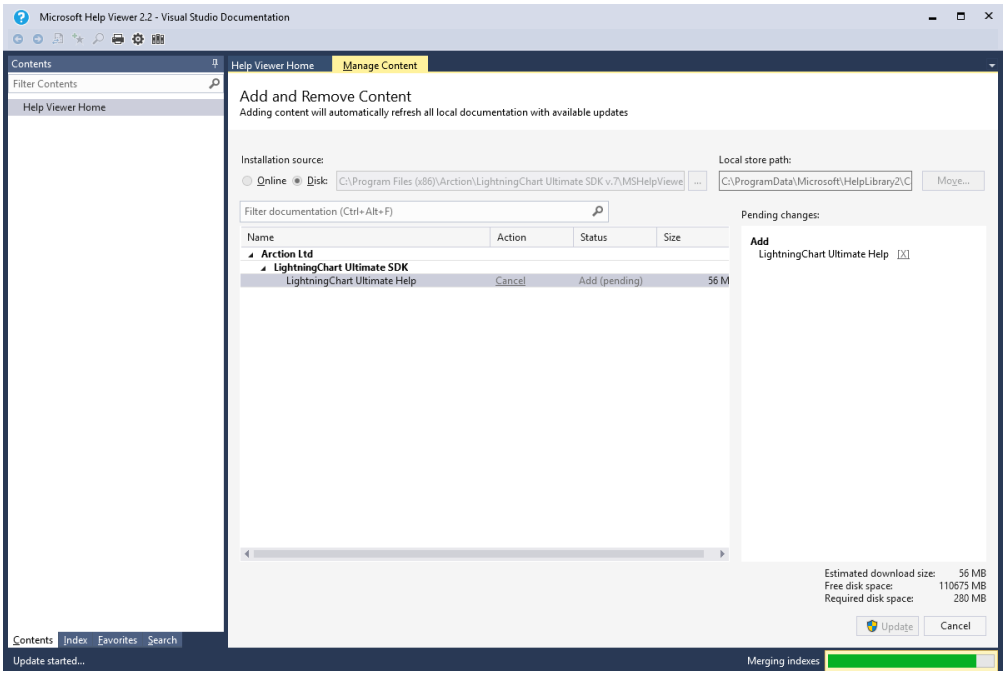
次の手順に従い、Visual Studio 2010 に LightningChart®.NET ヘルプコンテンツを手動でインストールします。

1. Visual Studio 2010 を開きます。
2. [*ヘルプ*]→ [*ヘルプ設定の管理*] を選択します。
3. ヘルプライブラリマネージャーで、[*設定*]リンクをクリックします。
4. [*ローカルヘルプを使用する*] が選択されていることを確認します。
5. [*ローカルヘルプを使用する*] が選択されている場合、[*キャンセル*]をクリックしてヘルプライブラリマネージャーに戻ります。それ以外の場合は、[***OK***]をクリックします。
6. [*ディスクリンクからコンテンツをインストール*] をクリックします。
7. [*参照*]ボタンをクリックし、LightningChart®.NET がインストールされているフォルダに移動します。デフォルトでは、***C:\Program Files (x86) \Arction\LightningChart .NET SDK v.10 \MSHelpViewer*** です。
8. ***HelpContentSetup.msha*** を選択し、[*開く*] ボタンをクリックします。
9. [*次へ*] ボタンをクリックします。
10. LightningChart®.NET Help の横に[*追加*]リンクがあります。それをクリックし、[*ステータス*]列の値が[*保留中の更新*] に変更されることを確認します。
11. [*更新*]ボタンをクリックします。ヘルプライブラリマネージャーが続行するかどうかを尋ねてきたら、[*はい*]ボタンをクリックします。ヘルプライブラリの更新が開始されます。
12. ヘルプライブラリが更新されたら、[*完了*]ボタンをクリックしてヘルプライブラリマネージャーを閉じます。

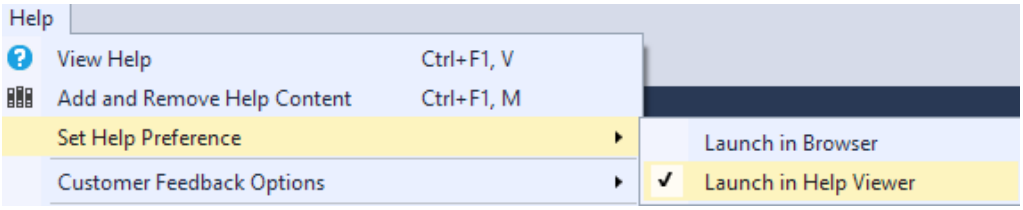
Visual Studio 2012-2019

次の手順に従い、Visual Studio 2012-2019 に LightningChart®.NET ヘルプコンテンツを手動でインストールします。

1. Visual Studio 2012、2013、2015、2017 または 2019 を開きます。
2. [*ヘルプ*]→[*ヘルプコンテンツの追加と削除*] を選択します。
3. Microsoft Help Viewer が起動したら、[*コンテンツの管理*] を選択します。
4. [*インストールソース*]の下の[*ディスク*]を選択します。
5. ファイルを参照するには、3 つの点があるボタンをクリックします。
6. LightningChart®.NET がインストールされているフォルダに移動します。デフォルトでは、パスは ***C:\Program Files (x86) \Arction\LightningChart .NET SDK v.10\MSHelpViewer*** です。
7. ***HelpContentSetup.msha*** を選択し、[*開く*] ボタンをクリックします。
8. LightningChart®.NET Help の横に[*追加*]リンクがあります。それをクリックし、[*ステータス*]列の値が[*保留中に追加*] に変更されることを確認します。



9. [*更新*]ボタンをクリックします。ヘルプライブラリマネージャーが続行するかどうかを尋ねてきたら、[*はい*]ボタンをクリックします。ヘルプライブラリの更新が開始されます。
10. ヘルプライブラリが更新されたら Microsoft Help Viewer を閉じることができます。
11. Visual Studio のメニュー/ヘルプで、[*ヘルプ設定：ヘルプビューアーで起動*] を選択します。



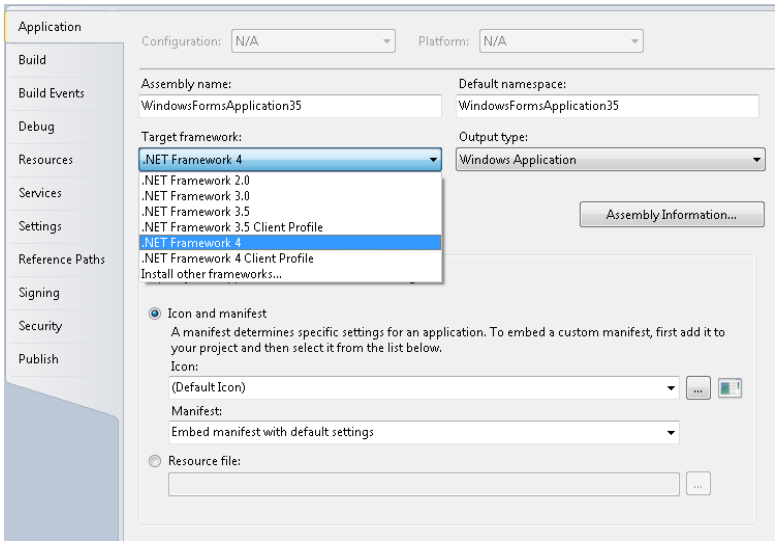
2.3 Visual Studio IntelliSense によるコードのパラメーターとヒント

LightningChartUltimate.dll ファイルがグローバルアセンブリキャッシュから参照されていて、コントロールが自動ツールボックスインストーラーによってインストールされていない場合、IntelliSense は LightningChart 関連コードを入力するときにコードヒントが表示されない場合があります。プロジェクトの参照リストから LightningChartUltimate.dll ファイルを削除します。次に、インストールディレクトリ（通常は ***C:\Program files (x86)\Arction\LightningChart .NET SDK v.10\LibNet4***）から参照して再度追加します。

2.4 ターゲットフレームワークの選択

C#プロジェクトでは、**[プロジェクト]→[プロパティ]→[アプリケーション]→[ターゲットフレームワーク]**でフレームワークを選択できます。

Visual Basic プロジェクトでは、**[プロジェクト]→[オプション]→[コンパイル]→[高度なコンパイルオプション]→[ターゲットフレームワーク]**でフレームワークを選択できます。



.NET Framework 4 Client Profile、または***.NET Framework 4***、***NET Framework 4.5*** 以上の物を選択します。

正しい.NET フレームワークが選択されている場合にのみ、LightningChart®.NET SDK コントロールが Visual Studio ツールボックスに表示されます。

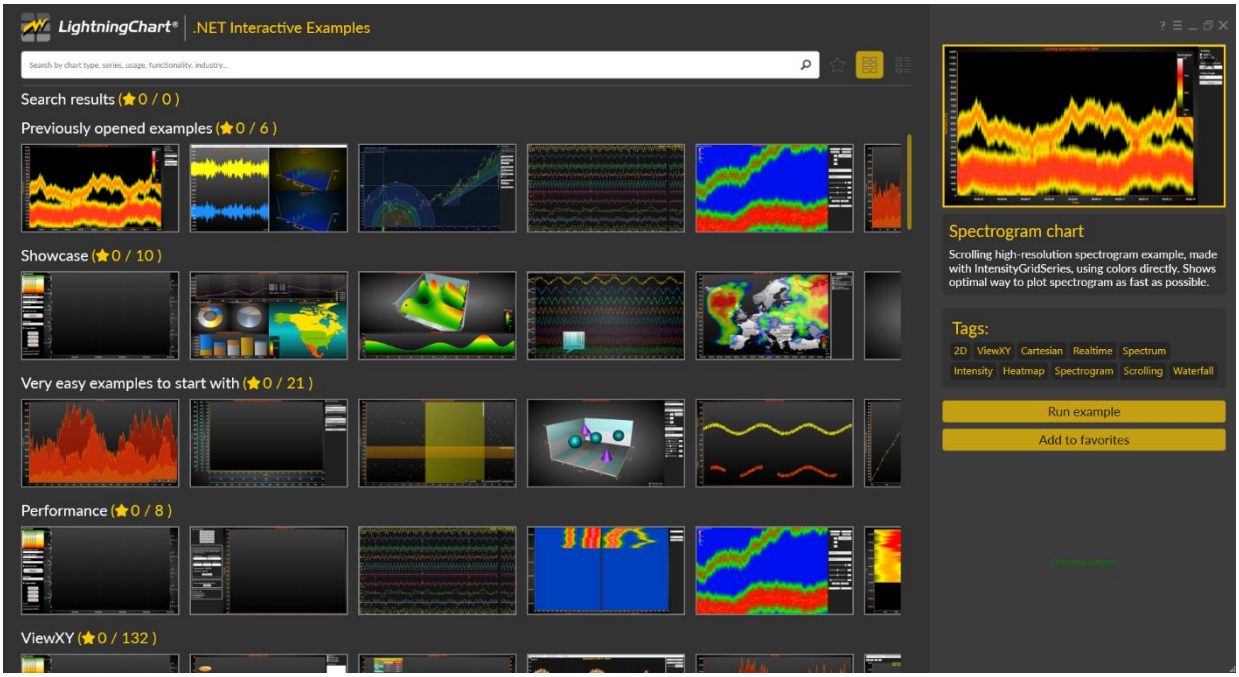
3. Dev Center

LightningChart .NET バージョン 8.5 以降では、**LightningChart .NET SDK v10.exe** セットアップを実行中に DevCenter が自動的にインストールされます。DevCenter は、LightningChart®.NET の機能とリソースにすばやくアクセスできる新しいアプリケーションです。マウスを数回クリックするだけで、次のタスクを実行できます。

- デモアプリケーションを開く
- チュートリアルや取扱説明書などのドキュメントリソースを開く
- メールでサポートに連絡する
- 技術サポートに送信できるアプリケーション情報を自動的に収集します。これにより、サポートチームがより迅速に問題解決するのに役立ちます。
- Arction Ltd.にフィードバックを送信するためのクイックリンク
- ライセンスステータスを確認し、ライセンスマネージャーを開いてライセンスを更新またはアクティブにする
- 新しいライセンスを購入する

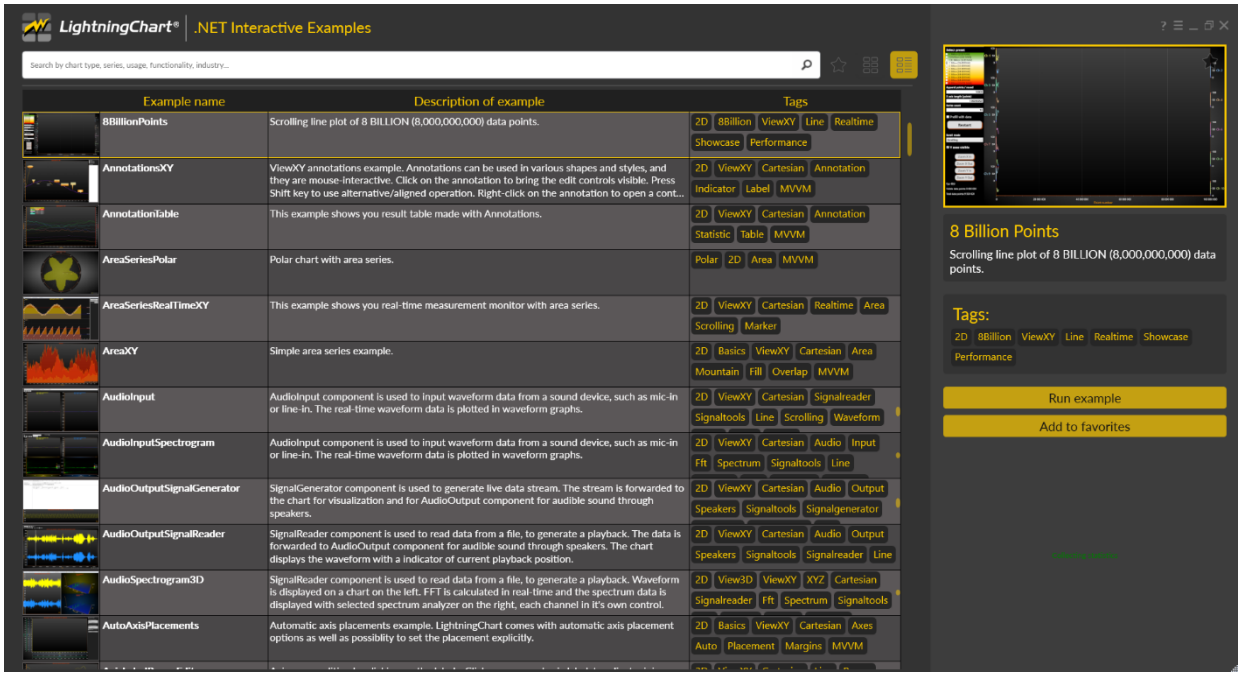
3.1 Interactive Examples を開く

デモアプリケーションは、LightningChart のコンポーネントと機能の使用方法を学習する際の主要な情報源の 1 つです。**デモアプリケーション**は、Dev Center のイメージの「デモアプリケーションを開く」をクリックするか、スタートメニューのショートカットを経由して実行する事ができます。デモアプリケーションには、様々なカテゴリーにグループ化された多くの例と、特定の例を探す検索バーがあります。さらに、個々のデモには**プロパティ**タブが付いており、チャートプロパティの修正を可能にしてくれます。

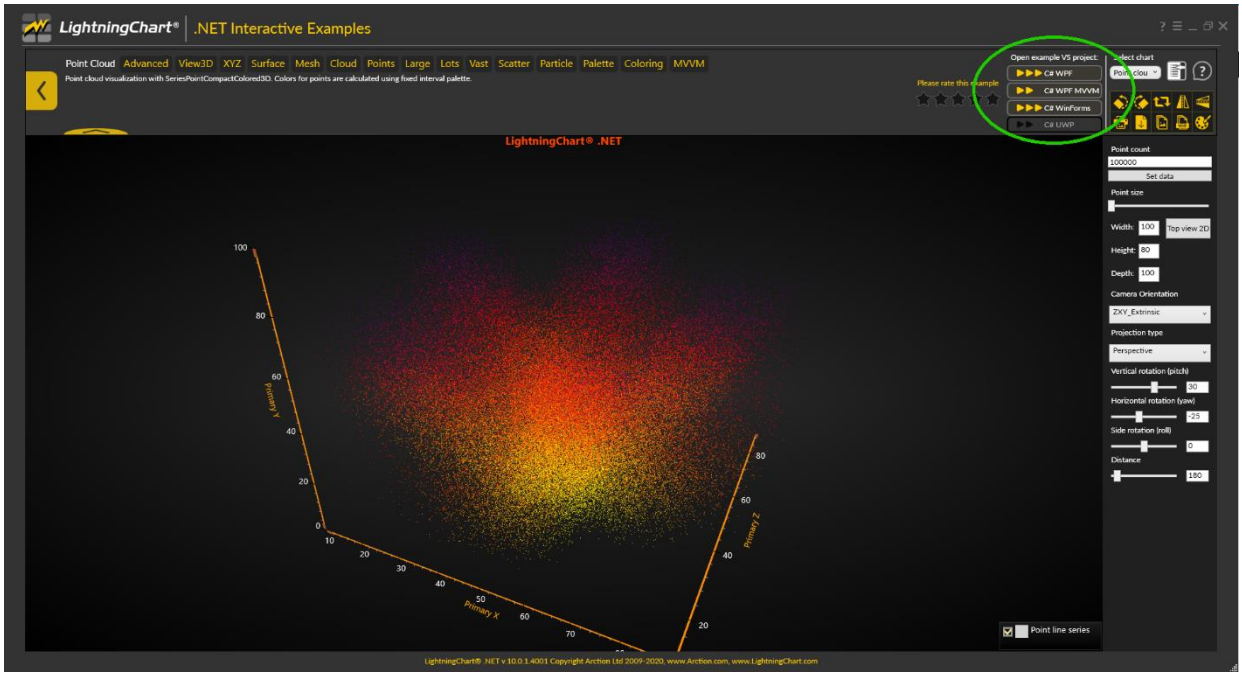


デモアプリケーションの Tile ビューは、カテゴリー毎に例を検索する事が可能です。

LightningChart .NET SDK をインストールすると、すべてのデモ例のソースコードがコンピューターに自動的に追加されます。デモアプリケーションの「プロジェクト毎の例を開く」のセクションをクリックすると、Visual Studio で独立型のプロジェクトとして、現行の例を開き、修正をする事ができます。



*表示のリスト化*で、カテゴリー無しですべての例が表示させます。



例が開いた状態。ハイライトの部分のボタンを通して、独立プロジェクトとして抽出する事ができます。

4.ライセンス管理

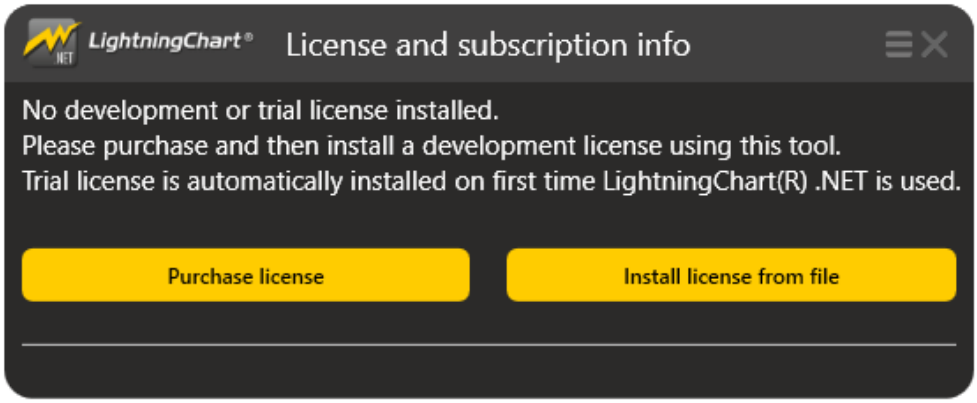
4.1 ライセンスを追加する

DevCenter または Windows のスタートメニュー：プログラム/ Arction / LightningChart .NET SDK / **License Manager** から License Manager アプリケーションを実行し、ライセンスを管理します。

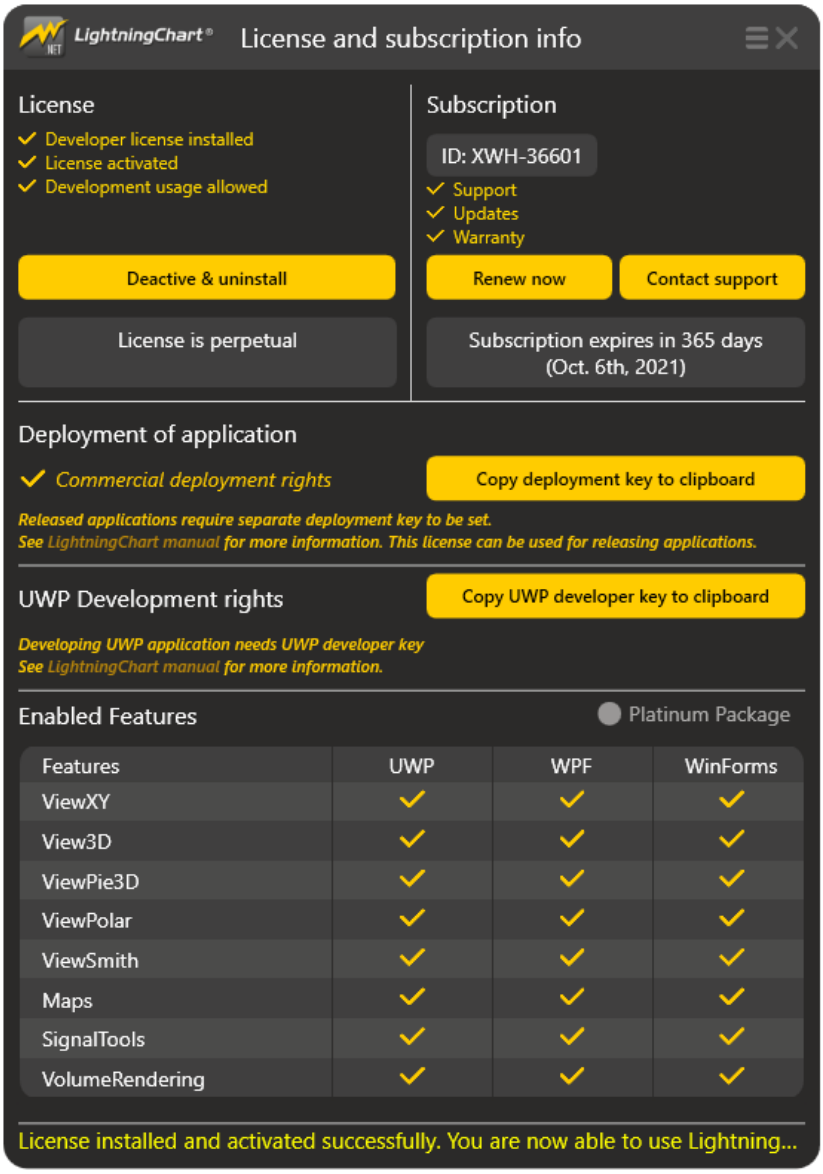
Arction コンポーネントは、ライセンスキー保護システムを使用します。コンポーネントは、有効なライセンスでのみ使用できます。ライセンスには次の情報が含まれます。

- ViewXY、View3D、ViewPie3D、Maps、ViewPolar、ViewSmith、ボリュームレンダリング シグナルツールなどの有効な機能
- WPF / WinForms /UWP/すべてのプラットホーム
- ライセンスをアクティベートできるユーザー数（標準で 1 人）
- サブスクリプションの有効期限（更新およびサポート終了日）
- 技術サポートの包括性
- 開発者ごとのライセンスまたはフローティングライセンス
- 学生ライセンス

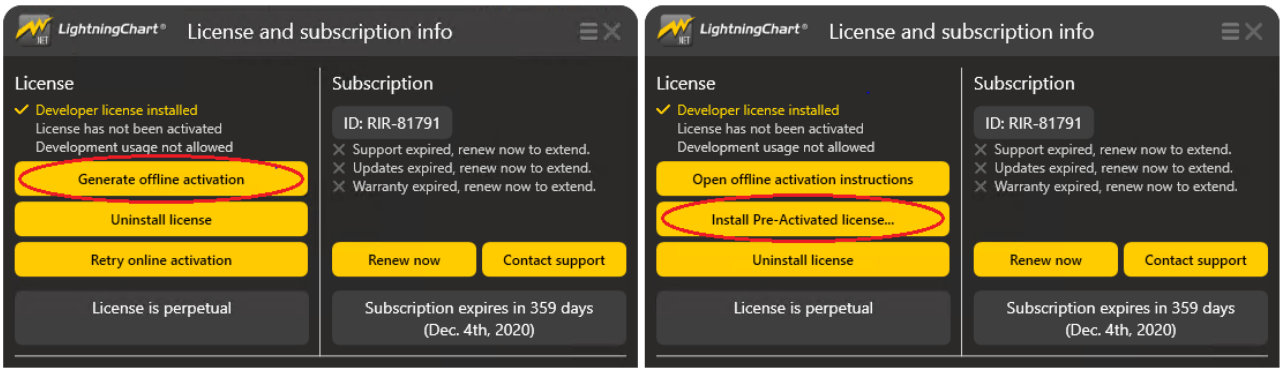
ツールボックスからアプリケーションに初めて Arction コンポーネントをドラッグすると、ライセンスマネージャーウィンドウでライセンスキーが要求されます。受信したライセンスファイルから、すべてのライセンスキーを一度に追加します。[**ファイルからライセンスをインストール…**]をクリックして、**.alf** ファイルを参照します。



開発者ごとのライセンスは、ライセンスファイルを追加した後、インターネット経由で Arction License Server に対して自動的にアクティベートされます。



インターネット接続が利用できない、または接続が遅すぎるなどの理由でオンラインアクティベーションが不可能な場合、ライセンスは電子メールからもアクティベートできます。それぞれのオンラインアクションが 1〜2 回失敗した後に、**[オフラインアクティベーションのリクエスト]**ボタンが使用可能になります。ライセンスをオフラインで無効化する方法も同様です。



オフラインボタンをクリックすると、画面上に指示が表示されます。それに従い、licensing@arction.com の Arction ライセンスチームにメールを送信します。Arction がライセンスをオフラインでインストールする方法を提供します。2 営業日以内に返信があります。

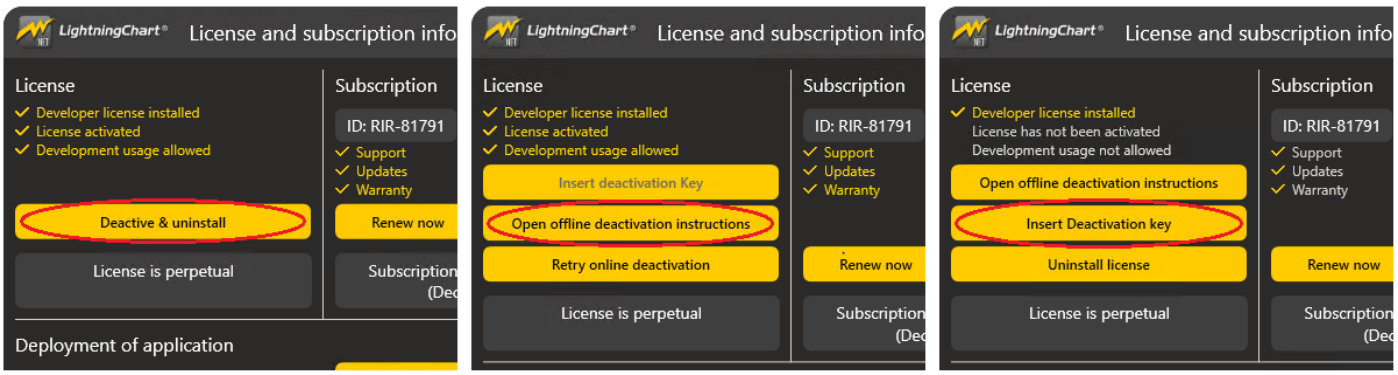
注意　キーコードには数千の文字が含まれているため、電話でのアクティベーション/アクティベーション解除はできません。

注意：LightningChart v.8.0 以降では、LIC フォーマットライセンスキーはサポートされておりません。ALF ライセンスは必要ありません。もしも ALF ライセンスをまだ入手していない場合は、Arction へ連絡してください。

4.2 ライセンスの削除

ライセンスは、**[アクティベート解除とアンインストール]**ボタンによってシステムから削除できます。自動でアクティベート解除するにはオンライン接続が必要です。インターネットに接続できない場合は、前章の通りメールでアクティベート解除できます。

ライセンスをアクティベート解除した後、別のコンピューターにインストールできます。

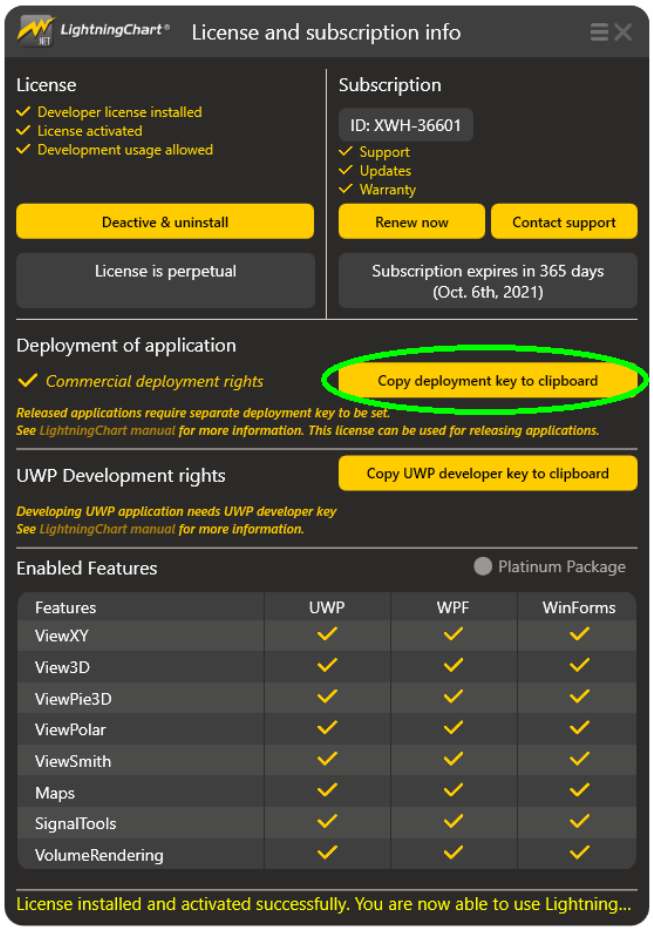


4.3 ライセンスの更新

ライセンスの初回インストール後は、例えば、購入期間が延長された場合、上級版にアップグレードされた場合、ソースコードが購入された場合などにおいては、さらに更新する事が可能です。ライセンスは、ユーザーの機器で自動的に**アップデートされない**事を覚えておいてください。よって、各ユーザーは、開発者のコンピューターでのライセンスが最新の物である事を確認するべきです。そのためには、古いライセンスは、まずは無効化され、削除されなければなりません。(前章の「ライセンスの削除はどうやっておこないますか?」を参照。) その後、新しいライセンスキー (.alf file) を Arction のカスタマーポータルから入手してください。それから、4.1 章の「ライセンスを追加する」に従ってインストールしてください。

4.4 展開キーの抽出

ソフトウェアが展開されるコンピューターで LightningChart アプリケーションを実行できるようにするには、展開キーをコードに適用する必要があります。展開キーは、**[展開キーをクリップボードにコピー]**ボタンを押すと、ライセンスキーから抽出できます。



4.5 アプリケーションでの展開キーの適用

コードでは、目的のコンポーネントに対して静的な ***SetDeploymentKey*** メソッドを使用します。使用されていないコンポーネントのキーを設定する（バインド不可のアプリケーションでバインド可能なチャートのキーを設定するなど）必要はありません。コンポーネントを使用する必要がある前に、***SetDeploymentKey*** メソッドを呼び出します。呼び出すのに最適な場所は、チャートを使用するクラスの*静的コンストラクター*か、アプリケーションのメインクラスです。

展開の詳細な手順については、第 5 章を参照してください。

WinForms

すべての WinForms アプリケーションに対してデフォルトで作成される *Program* クラスの静的コンストラクターメソッドでキーを適用する方法の例を示します。

```
namespace WindowsFormsApplication1
{
   static class Program
   {

        static Program()
        {
  //Set Deployment Key for Arction components
  string deploymentKey = "VMalgCAAO6kO1RgiNIBJABVcG.R..Kikfd...";

  Arction.WinForms.Charting.LightningChart.SetDeploymentKey(deploymentKey);
  Arction.WinForms.SignalProcessing.SignalGenerator.SetDeploymentKey(deploymentKey);
  Arction.WinForms.SignalProcessing.AudioInput.SetDeploymentKey(deploymentKey);
  Arction.WinForms.SignalProcessing.AudioOutput.SetDeploymentKey(deploymentKey);
  Arction.WinForms.SignalProcessing.SpectrumCalculator.SetDeploymentKey(deploymentKey);
  Arction.WinForms.SignalProcessing.SignalReader.SetDeploymentKey(deploymentKey);
        }

        // Rest of the class ...
   }
}
```

WPF

App クラスの静的コンストラクターで、App.xaml.cs の先頭にキーを適用する方法の例を示します。

```
using Arction.Wpf.SignalProcessing;

namespace WpfApplication1
{
   /// <summary>
   /// Interaction logic for App.xaml
   /// </summary>
   public partial class App : Application
   {
            static App()
            {
```

```
            // Set Deployment Key for Arction components
            string deploymentKey = "- DEPLOYMENT KEY FROM LICENSE MANAGER
            GOES HERE-";

            // Setting Deployment Key for bindable chart
            Arction.Wpf.ChartingMVVM.LightningChart
            .SetDeploymentKey(deploymentKey);

            // Setting Deployment Key for non-bindable chart
            Arction.Wpf.Charting.LightningChart
            .SetDeploymentKey(deploymentKey);

            // Setting of deployment key to other Arction components
            SignalGenerator.SetDeploymentKey(deploymentKey);
            AudioInput.SetDeploymentKey(deploymentKey);
            AudioOutput.SetDeploymentKey(deploymentKey);
            SpectrumCalculator.SetDeploymentKey(deploymentKey);
            SignalReader.SetDeploymentKey(deploymentKey);

        }
    }
```

UWP アプリケーションでは、開発者キー、あるいは展開キーのいずれかを使用する事が可能です。しかし、両方を一緒に使用する事は出来ません。開発中およびアプリのでバッギングの際は、開発者キーを、展開中は展開キーを使ってください。
注意：アプリケーションに展開キーを設定しないと、LightningChart アプリケーションはターゲットマシンで 30 日間の試用モードに入ります（開発ライセンスキーがインストールされていないコンピューターに適用されます）。

4.6 開発コンピューターで展開キーを使用し実行する

開発ライセンスがインストールされているコンピューターで、展開キーが ***SetDeploymentKey*** で適用されているアプリケーションを実行する場合、**ライブラリは開発ライセンスキーを優先します**。展開キーに、ローカルにインストールされたライセンス（シルバーパックなど）よりも高いレベルの機能（ゴールドパックなど）が含まれている場合、ユーザーまたはデバッグの混乱につながる可能性があります。開発者はこの制限に注意する必要があります。

Arction は、すべてのライセンスをチーム全体で同じタイプにすることを推奨しています。

4.7 デバッガーで実行する

展開キーが正しく設定された状態でデバッガーが接続された Visual Studio からプロジェクトを実行し、システムから開発ライセンスキーが見つからない場合、チャートはスローレンダリ

ングモードになります。最大 FPS は〜1 であり、チャートにメッセージテキストが表示されます。

開発者ライセンスキーなしの LightningChart による直接開発とデバッグは、LightningChart EULA で禁止されています。

4.8 試用期間

試用期間は 30 日間です。その後製品の使用を継続するにはライセンスを購入する必要があります。トライアルライセンスでビルドされたすべてのプロジェクトは、適切なライセンスに更新した後でも機能します。試用版ライセンスで作成されたチャートアプリケーションを実行すると、試用版のナグメッセージが表示されます。

4.9 フローティングライセンス

フローティングライセンスは、無制限の数のコンピューターにインストールできます。同時開発者の数は、Arction によって構成されています。購入した同時ユーザーの数だけが、同時に LightningChart で開発できます。開発者が LightningChart の開発を終了すると、別の開発者が使用を開始できるようになるまで約 10〜15 分のタイムアウトがあります。

展開キーは、開発者ごとのライセンスと同様に設定する必要があります。 フローティングライセンスは、デフォルトで Arction Licensing Server によってコントロールされます。開発中は継続的なインターネット接続が必要です。

*顧客側のフローティングライセンスコントローラー*も利用できます。開発用コンピューターは、ローカルエリアネットワークを介して顧客の組織で実行されているサービスに接続します。Arction または他の関係者とのオンライン通信は行われません。ライセンスでは、Arction はコントローラーサービスとフローティングライセンスをインストールするための個別の指示を提供します。

5. LightningChart アセンブリの展開/配布

5.1 参照アセンブリ

Arction .dll ファイルを、実行可能フォルダー、実行可能フォルダーの隣、グローバルアセンブリキャッシュ、または.NET アセンブリ解決システムがそれらを見つけることができる別のフォルダーで配信します。LightningChart は ***ClickOnce*** 展開もサポートしています。

WinForms:

- Arction.WinForms.Charting.LightningChart.dll
- Arction.Licensing.dll
- Arction.DirectX.dll
- Arction.RenderingDefinitions.dll
- Arction.RenderingEngine.dll
- Arction.RenderingEngine9.dll
- Arction.RenderingEngine11.dll
- Arction.DirectXInit.dll
- Arction.DirectXFiles.dll

SignalTools を使用している場合

- Arction.WinForms.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

WPF:

- Arction.Wpf.Charting.LightningChart.dll **(バインド不可の WPF chart 用)**
- Arction.Wpf.ChartingMVVM.LightningChart.dll **(バインド可能の WPF chart 用)**
- Arction.Licensing.dll
- Arction.DirectX.dll
- Arction.RenderingDefinitions.dll
- Arction.RenderingEngine.dll
- Arction.RenderingEngine9.dll
- Arction.RenderingEngine11.dll
- Arction.DirectXInit.dll
- Arction.DirectXFiles.dll

SignalTools を使用している場合

- Arction.Wpf.SignalProcessing.SignalTools.dll
- Arction.MathCore.dll

UWP:

- Arction.Uwp.ChartingMVVM.LightningChart.dll
- Arction.Uwp.RenderingDefinitions.dll
- Arction.Uwp.RenderingEngine.dll
- Arction.Uwp.RenderingEngineBase.dll
- Arction.Uwp.Licensing.dll
- SharpDX.D3DCompiler.dll
- SharpDX.Direct2D1.dll
- SharpDX.Direct3D11.dll
- SharpDX.dll
- SharpDX.DXGI.dll
- SharpDX.Mathematics.dll
- UwpAttributes.dll

コンポーネントすべてに、***SetDeploymentKey*** メソッドを使用してください。そうでない場合、チャートは試用モードに入り、30 日間のみ動作して試用ナグが表示されます。
DeploymentKey および詳細なライセンスキー管理の作成については、第 4 章を参照してください。

5.2 その他のコメント

LightningChart のライセンスキーが.NET 逆アセンブラツールに表示されないように、アプリケーションコードを難読化することは**必須**です。ライセンスキーの漏洩は、ライセンスの終了、法的措置、損害賠償の請求につながる可能性があります。

LightningChart ソースコードサブスクライバーは、LightningChart ライブラリのソースコードにアクセスします。Arction の知的財産権とコードの漏洩を防ぐために、LightningChart ソースコードからビルドされたアセンブリを難読化することは**必須**です。難読化されていない LightningChart ライブラリの配布は EULA に違反し、ライセンスの終了、法的措置、損害賠償の請求につながる可能性があります。

これらの XML ファイルの展開は禁止されています。

- Arction.WinForms.Charting.LightningChart.xml
- Arction.Wpf.BindableCharting.LightningChart.xml
- Arction.Wpf.Charting.LightningChart.xml
- Arction.Wpf.ChartingMVVM.LightningChart.xml
- Arction.Wpf.SignalProcessing.SignalTools.xml

- Arction.WinForms.SignalProcessing.SignalTools.xml

Arction が提供するファイルは、アプリケーション開発の支援のみを目的としています。主にコードパラメータとプロパティのアドバイスを表示するために使用されます。ソースコードから LightningChart アセンブリを再構築する場合、上記の XML ファイルが展開されていないことを確認してください。.NET 逆アセンブラーおよびリバースエンジニアリングアプリケーションの情報が多すぎるため、**それらを配布することは固く禁じられています。**

6. LightningChart コンポーネント

6.1 LightningChart® .NET ライブラリの使用

LightningChart®.NET コンポーネントを使用するには、Arction .dll ファイルを参照に追加する必要があります。これらはインストールフォルダにあります。アプリケーションの開発時には、以下のアセンブリが必要です。

Winforms:	Arction.WinForms.Charting.LightningChart.dll.
WPF バインド不可:	Arction.Wpf.Charting.LightningChart.dll Arction.DirectX.dll Arction.RenderingDefinitions.dll
WPF バインド可能:	Arction.Wpf.ChartingMVVM.LightningChart.dll Arction.DirectX.dll Arction.RenderingDefinitions.dll
UWP Chart:	Arction.Uwp.ChartingMVVM.LightningChart.dll Arction.Uwp.RenderingDefinitions.dll Arction.Uwp.RenderingEngineBase.dll

SignalTools を使用する場合: Arction.WinForms.SignalProcessing.SignalTools.dll または、
Arction.Wpf.SignalProcessing.SignalTools.dll または、
Arction.Uwp.SignalProcessing.SignalTools.dll

上記の参照が追加されると、プロジェクトをビルドする時に必要なすべてのアセンブリが出力フォルダーに自動的にコピーされます。第 28 章では、LightningChart アプリケーションを展開するときに必要なアセンブリを説明します。

Arction.DirectXFiles.dll は初期化時間を延長させる可能性がある大きなファイルであるため、 自動的には参照として含まれません。システムに正しい DirectX アセンブリが既に存在しない場

合にのみ必要になります。Arction.DirectXInit.dll ルーチンは、既存の dll をチェックして必要に応じてそれらをロードします。一度ロードされると DirectX-dll が Windows の一時フォルダーに書き込まれ、LightningChart は今後そこにアクセスできるようになり、初期化が高速になります。

Arction.DirectXFiles.dll を参照に含めず、exe の横にコピーすることを推奨します。

6.2 コードでチャートを作成する

LightningChartUltimate コンポーネントを追加するには、ツールボックスからドラッグするか、完全にコードビハインドで作成します。コードでチャートオブジェクトを作成すると、バージョンの更新が簡単になるという利点があります。さらに、（非）シリアル化に関連する問題を回避できます。

以下は、コードビハインド（.xaml.cs -file）で WPF バインド不可のチャートを作成する方法の一つです。

```
using Arction.Wpf.Charting;

namespace ExampleProject
{
    public partial class ExampleApp : Page
    {
        private LightningChart _chart = null;

        public ExampleApp()
        {
            InitializeComponent();

            CreateChart();
        }

        private void CreateChart()
        {
            _chart = new LightningChart();

            // Chart control into the parent container.
            (Content as Grid).Children.Add(_chart);

            // Disable rendering until the whole chart is set up correctly.
            _chart.BeginUpdate();

            // Configure chart here.

            // Allow rendering the chart.
            _chart.EndUpdate();
        }
    }
```

```
}
```

6.3 ツールボックスからチャートを追加する

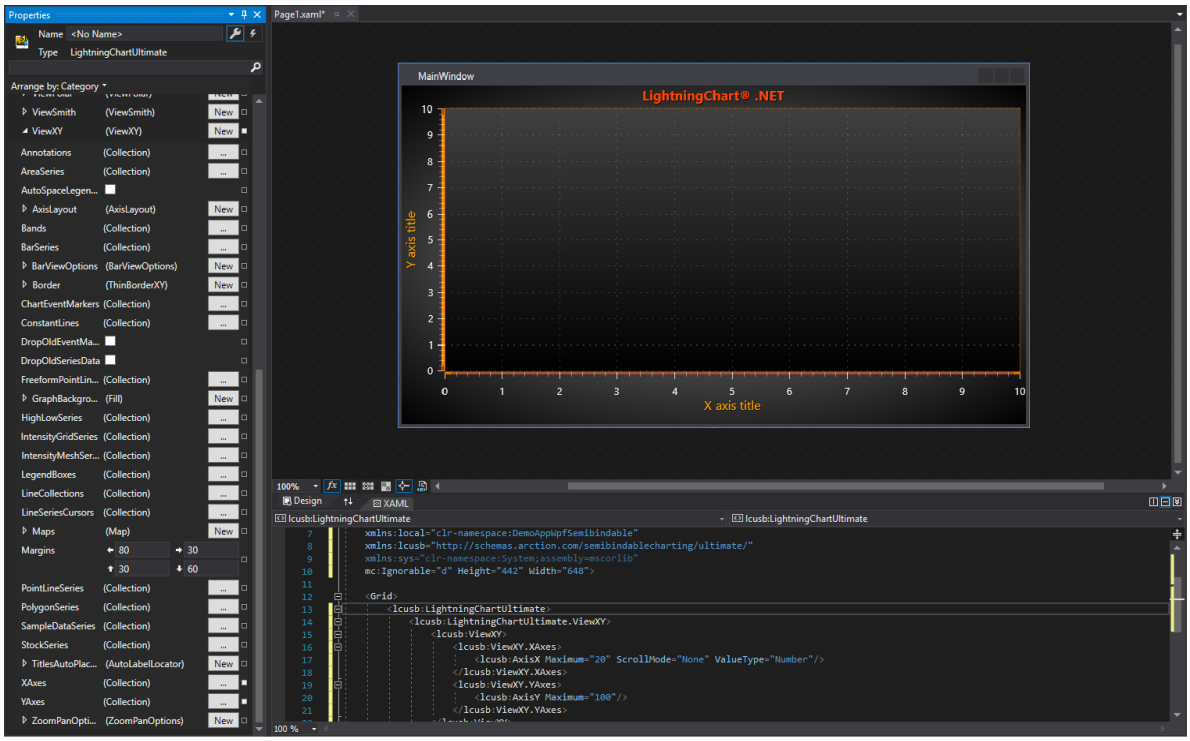
ツールボックスからフォームに ***LightningChart*** コントロールを追加します。チャートがフォームに表示され、そのプロパティが**プロパティウィンドウ**に表示されます。バインド可能な WPF を試用している場合は、XAML エディターは、チャートのデフォルトプロパティの内容と変更を表示します。

プロパティは自由に変更でき、新しいシリーズや他のオブジェクトをコレクションに挿入できます。 系列データポイントはコードで指定する必要があります。チャートのメインレベルのイベントハンドラーは、プロパティグリッドで割り当てることができます。コレクションに追加されたオブジェクトの場合、イベントハンドラーをコードで割り当てる必要があります。

バージョン更新に関するベストプラクティス

チャートプロパティデータは、Visual studio プロジェクトの ***.resx*** ファイルにシリアル化されます。LightningChartUltimate API はバージョンの更新によって少し変更される傾向があり、新しいバージョンが**.resx** ファイルで機能するために、互換性のないシリアル化がされる場合があります。

バージョンを簡単に更新するには、コードでチャートオブジェクトを作成し、すべてのシリーズ、イベントハンドラーなどを追加することを推奨します。そうした場合プロジェクトは正しく読み込まれ、起こりうるエラーがコンパイル時に表示されるため、.resx ファイルを修正するよりも簡単に修正できます。.resx ファイルでは、一部のプロパティ定義が失われる場合がありますが、コードでは常に指定されます。



6.4 UWP プロジェクトを作成する

UWP での LightningChart の使用は、同じようなバインディングと MVVM 能力を持っているため、バインド可能な WPF チャートと同じような働きをします。バインド可能な WPF チャートを使用する時のように、UWP チャートのコレクションプロパティ (ViewXY 軸、3Dlights など) は、デフォルトで空になります。LightningChart で UWP アプリケーションを開発するためには、 Windows10, Visual Studio 2017 あるいは 2019 が必要です。 また Universal Windows Platformn の開発ワークロードも下記の条件で、Visual Studio でインストールする必要があります。

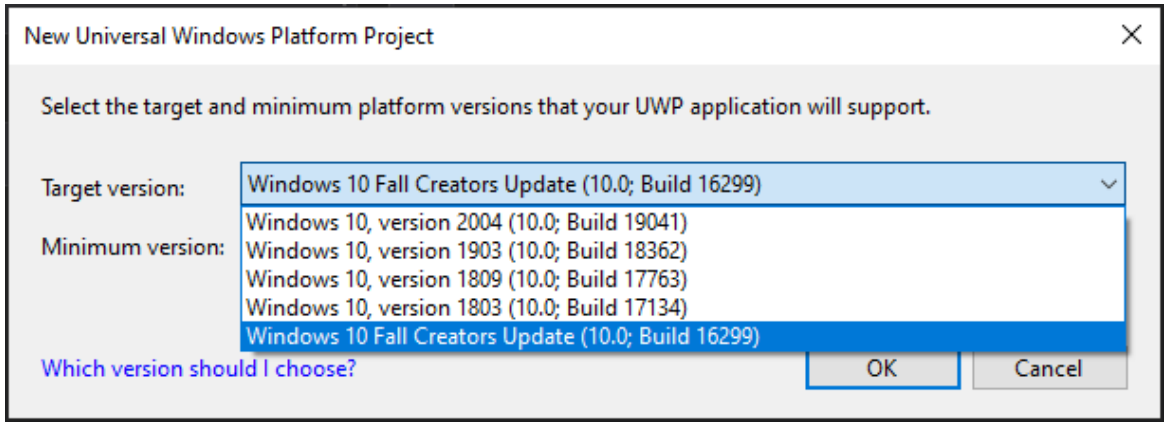
- Microsoft.NETCore.UniversalWindowsPlatform: 6.2.8 または、それ以降の物 (Nuget パッケージ).
- Microsoft.Toolkit.Uwp: v 4.0.0 または、それ以降の物、 6.0.0 またはそれ以降の物をお勧めします。 最新版のツールキットは、それ以前の対象バージョンとは互換性が無い場合があります。

UWP アプリケーションを作成する

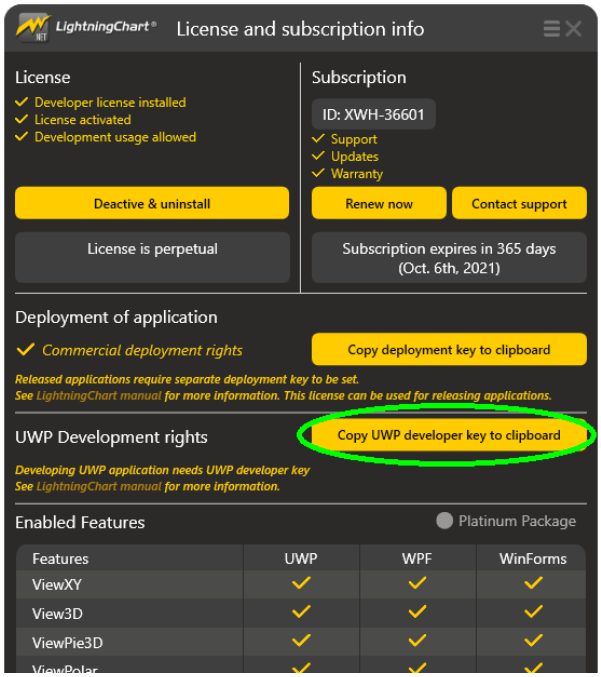
LightningChart を使って UWP アプリケーションを作成する場合は、下記の手順に従ってください。

1. Visual Studio で新しいプロジェクトを作成する。 ***Blank App (Universal Windows)*** を選択する。

2. プロジェクト名を付け、ファイルロケーションを決める。
3. プロジェクト用に ***Target*** と ***Minimum*** バージョンを設定する。どのバージョンが利用できるかは、機器にインストールされた SDK によります。詳細情報については、マイクロソフトの資料 https://docs.microsoft.com/en-us/windows/uwp/updates-and-versions/choose-a-uwp-version を参照してください。16299 以降のバージョンをお勧めいたします。尚、バージョンについては、Project ->Properties を経由して後で変更する事ができます。



4. LightningChart アセンブリをレファレンス ***Arction.Uwp.ChartingMVVM.dll***, ***Arction.Uwp.RenderingDefinitions.dll*** および ***Arction.Uwp.RenderingEngineBase.dll*** に追加してください。 同じ UWP アセンブリは、 x86, x64, Arm and Arm64 のプラットホームに使用できます。ターゲットプラットホームは、プロジェクトを右クリックし、*Properties -> Build -> Platform target* で選択をして変更できます。
5. プロジェクトへ ***Microsoft.Toolkit.Uwp*** NuGet パッケージをインストールします。6.0 以降のバージョンをお勧めします。
6. アプリケーションで設定するには、UWP は開発者キーが必要です。***Copy UWP developer key to clipboard***" ボタンを押して LicenseManager からキーを抽出してください。その後、***LightningChart.SetUwpDeveloperKey()*** 方式を使って、***App.Xaml.cs*** ファイルに設定してください。




```
using Arction.Uwp.ChartingMVVM;

namespace ExampleProject
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    1 reference
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        0 references
        public App()
        {
            LightningChart.SetUwpDeveloperKey("Copy developer key here");

            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }
```

7. この時点で、LightningChart コンポーネントをコードの中で、あるいはツールボックスからコードをドラッグして作成する事が可能です。
8. アプリケーションを組み立て、展開、実行してください。アプリが実行されない場合(例、「Windows ストアアップのアクティベーション...」によるエラー)は、Target と Minimum バージョンを変更すると多くの場合有効です。
9. UWP を他の機器に対して展開する場合、展開キーの適用が必要です。(第 4.4 章参照。) 展開キーは、開発キーと一緒使う事はできません。したがって、展開する前に開発者キーの設定を取り除いてください。

6.5 Windows フォーム、WPF 、UWP の違い

Windows フォームと WPF の間のプロパティツリーとオブジェクトモデルは、チャートカテゴリに関してほぼ同一です。主な違いは次のとおりです。

	Window フォーム	WPF	UWP
レンダリングオプションプロパティ	RenderOptions	ChartRenderOptions	ChartRenderOptions
背景塗りつぶしプロパティ	Background	ChartBackground	ChartBackground
フォント	System.Drawing.Font	Arction.WPF.LightningChart.WPFFont	Arction.Uwp.ChartingMVVM.UwpFont
カラー	System.Drawing.Color	System.Windows.Media.Color	Windows.UI.Color

6.6 LightningChart ビュー

LightningChart の主なビューは次の通りです。

- ViewXY (第 **Error! Reference source not found.**章を参照)

- View3D (第 0 章を参照)

- ViewPie3D (第 0 章を参照)

- ViewPolar (第 0 章を参照)

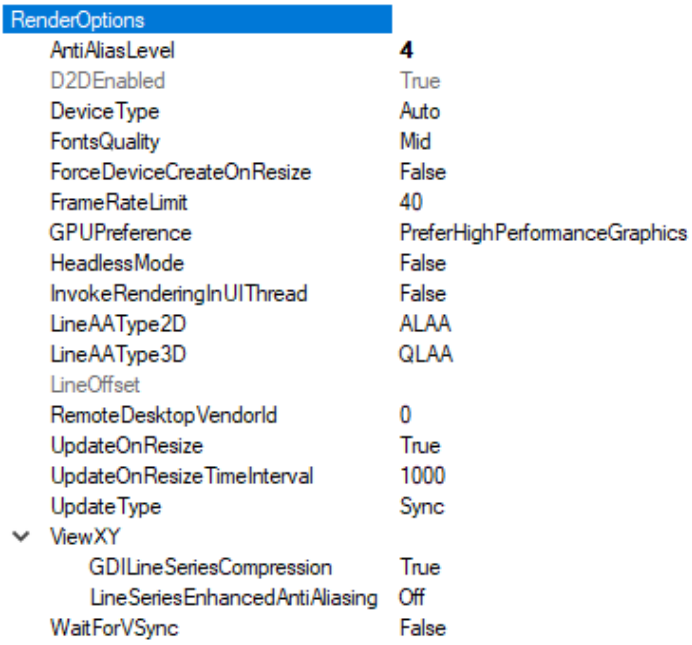
- ViewSmith (第 0 章を参照)

ActiveView プロパティを設定することにより、表示ビューを変更できます。 デフォルトのレビューは、ViewXY です。

```
// Set 3D as the visible view
chart.ActiveView = ActiveView.View3D;
```

6.7 外観/パフォーマンス設定の構成

ChartRenderOptions [WinForms の ***RenderOptions***] には、外観とパフォーマンスを構成するためのプロパティが含まれています。



DeviceType

```
// Changing the rendering device in code
chart.ChartRenderOptions.DeviceType = RendererDeviceType.Auto;
```

Auto は AutoPreferD11 オプションのエイリアスです。 これがデフォルト設定です。

AutoPreferD9 は DirectX9 ハードウェアレンダリングを優先し、可用性に基づいて HW9->HW11-> SW11-> SW9 の順にデバイスを自動的に選択します。ハードウェアが利用できない合、WARP [SW11] ソフトウェアレンダリングにフォールバックします。

AutoPreferD11 は、DirectX11 ハードウェアレンダリングを優先し、可用性に基づいて HW11-> HW9-> SW11-> SW9 の順にデバイスを自動的に選択します。ハードウェアが利用できない場 合、WARP [SW11] ソフトウェアレンダリングにフォールバックします。**これを最良の性能 と外観の一般的な設定として使用します。**外観は DirectX9 レンダラーよりも優れています。

HardwareOnlyD9 は、ハードウェア 9 レンダリングのみを使用します。

HardwareOnlyD11 は、ハードウェア 11 レンダリングのみを使用します。

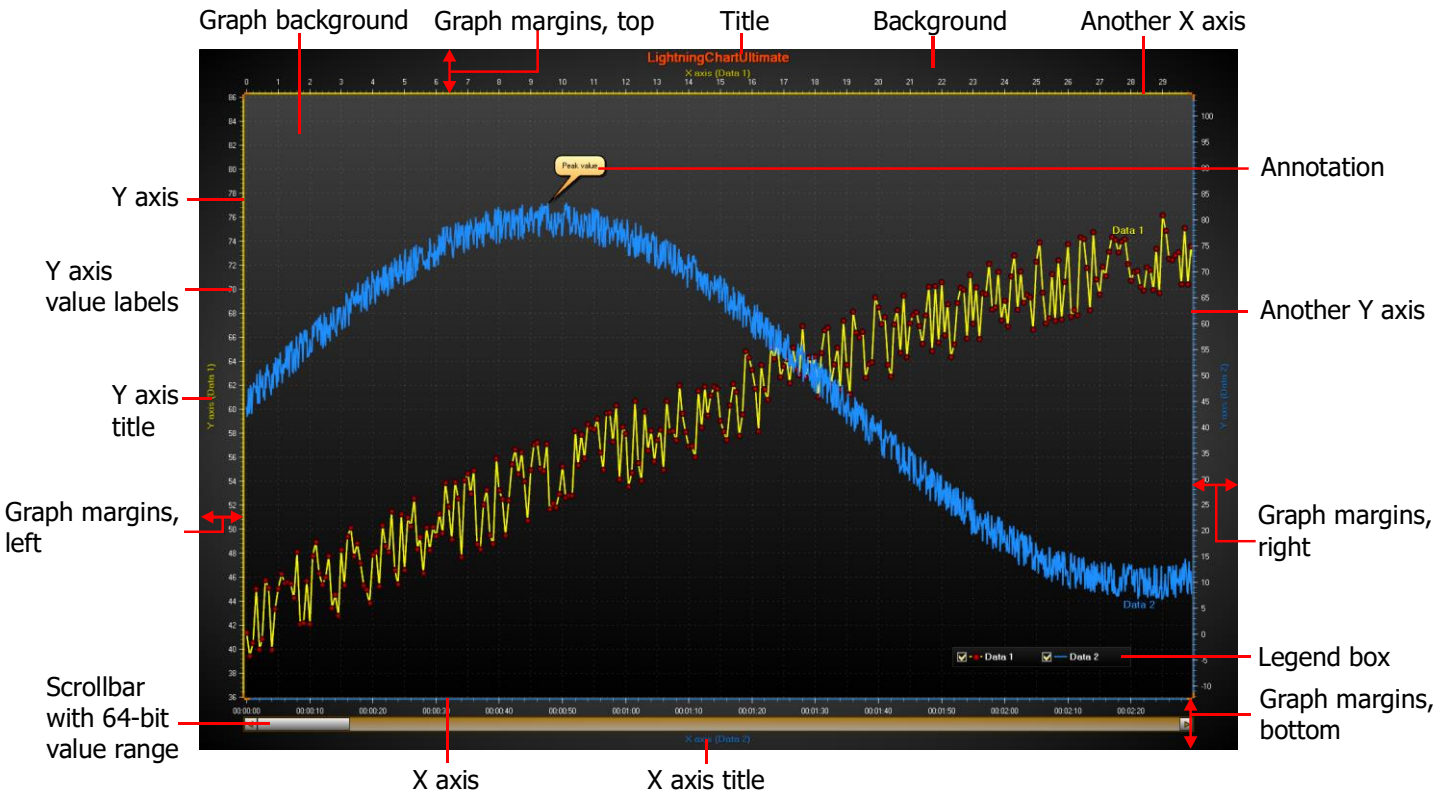
SoftwareOnlyD11 は DirectX11 WARP を使用します (DirectX9 参照ラスタライザーと比較すると 非常に高速ですが、ハードウェアオプションよりも低速です)。

SoftwareOnlyD9 は DirectX9 参照ラスタライザーを使用します（非常に遅い）。

None はチャートが非表示の場合、またはバックグラウンドで非アクティブの場合です。
DeviceType を ***None*** に設定すると、グラフィックリソースが他のチャートに解放されます。

7. ViewXY

ViewXY では、ポイントラインシリーズ、エリアシリーズ、ハイローシリーズ、ヒートマップ、バーシリーズ、バンド、ラインシリーズカーソル、地理マップなどの様々な二次元シリーズを Cartesian、XY グラフ形式で表示できます。シリーズは X 軸と Y 軸にバインドされ、割り当てられた軸の値の範囲を使用しています。



7.1 軸レイアウト

軸の配置、自動マージンなどを調整する一般的なプロパティは、***ViewXY.AxisLayout*** プロパティとサブプロパティにあります。

XAxisAutoPlacement は、X 軸を垂直に配置する方法をコントロールします

```
chart.ViewXY.AxisLayout.XAxisAutoPlacement = XAxisAutoPlacement.AllTop;
```

YAxisAutoPlacement は、Y 軸の水平方向の配置をコントロールします。

複数の Y 軸が定義されている場合、3 つの異なる方法 ***Layered***、***Stacked***、***Segmented*** で縦に並べることができます。これは ***ViewXY.AxisLayout.YAxesLayout*** プロパティで選択できます。

Layered

Layered では、すべての Y 軸はグラフの上部から始まり、グラフの下部まで伸びます。軸とそれらにバインドされたシリーズは、同じ垂直方向のスペースを共有します。

Stacked

Stacked では、各 Y 軸に独自の垂直方向のスペースがあります。すべての Y 軸の高さは同じです。

Segmented

Segmented では、垂直方向のスペースがセグメント間で分割されます。各セグメントには複数の Y 軸を含めることができます。各セグメントの関係の**高さを**設定でき、セグメント内のすべての Y 軸がセグメントの高さを取得します。

Segments は ***AxisLayout.Segments*** コレクションで作成する必要があります。最初に追加されたセグメントは、チャートの下部に配置されます。Y 軸は、***yAxis.SegmentIndex*** プロパティを設定することでセグメントに割り当てることができます。***SegmentIndex*** は、***AxisLayout.Segments*** コレクションのインデックスです。

7.1.1 範囲設定

Minimum プロパティと ***Maximum*** プロパティに値を指定して、軸の値の範囲を設定します。両方の値を同時に設定するには、***SetRange(…)*** メソッドを使用します。値の範囲は、***AllowScrolling*** が有効の場合、直接軸をマウスでドラッグしてスクロールする事ができます。***Minimum*** と ***Maximum*** は、***AllowScrolling*** プロパティが有効の場合は、スケールニブエリア (軸の最後の所) を上下にドラッグして変更する事ができます。

7.1.2 区分とグリッド

MajorDiv および ***MinorDiv*** プロパティによって設定される区分は、チャートの大目盛りおよび小目盛りの量を決定します。たとえば、5 つの主要な区分を設定すると、目盛りと主要なグリッド線で区切られた 5 つの等しいサイズのスペースに Y 軸が分割されます。デフォルトでは、大目盛りが有効、小目盛りが無効になっています。

AutoDivSpacing プロパティを使用すると、主要な部門を自動的に計算できます。これはデフォルトで有効になっています。間隔は、可能な限りユーザーフレンドリーになるように、値ラベルのフォントサイズと ***AutoDivSeparationPercent*** プロパティに基づいて計算されます。

AutoDivSpacing が無効になっている場合、***MajorDiv*** および ***MajorDivCount*** プロパティを使用して、分割間隔を手動で設定できます。***MajorDiv*** は大きさによって間隔を設定しますが、***MajorDivCount*** は分割数によって間隔を設定します。***KeepDivCountOnRangeChange*** プロパティを使用すると、***MajorDiv*** の設定に関係なく、軸の範囲が変更されるたびに分割カウントを強制的に維持できます。

大目盛りの目盛りスタイルは、***MajorDivTickStyle*** プロパティから設定できます。***MajorDivTickStyle.Alignment*** プロパティを使用して、目盛りとラベルの方向を編集します。値ラベルは、主要な目盛りの横に描画されます。小区分のプロパティはそれぞれ ***MinorDivTickStyle*** プロパティで変更できます。

水平グリッド線は、目盛りの垂直位置に描かれます。主要な目盛りには大きなグリッド、小目盛りには小グリッドが描かれます。***MajorGrid*** および ***MinorGrid*** プロパティを使用して、グリッドの外観を編集できます。

7.1.3 カスタム目盛り

軸の目盛りの位置とラベルテキストは、カスタムの目盛りを使用して手動で設定できます。***CustomTicksEnabled*** を true に設定し、***CustomTicks*** リストプロパティで目盛りの位置を定義します。カスタム目盛りは、Tick、Grid、またはその両方で構成できます。***Style*** を使用して、それぞれ Tick、Grid、または TickAndGrid を選択します。目盛りまたはグリッドの色は、***Color*** プロパティを介して変更できます。***Length*** プロパティで目盛りの長さを設定します。グリッド線のパターンは、軸の ***MajorGrid.Pattern*** および ***PatternScale*** プロパティの設定に従います。

CustomAxisTick には、位置と対応するラベルテキストを定義する ***AxisValue*** プロパティと ***LabelText*** プロパティがあります。カスタム目盛りを使用する場合、***AutoFormatLabels*** を無効にしてカスタムラベルテキストを表示します。さらに、コードで新しいカスタム目盛りを設定した後に ***InvalidateCustomTicks ()*** を呼び出す必要があります。

7.1.4 対数軸

対数プレゼンテーションを使用するには、***ScaleType*** を ***Logarithmic*** に設定します。***LogBase*** プロパティで対数ベース値を設定します。チャートには、0〜1 の対数値も表示できます。***LogZeroClamp*** を使用して、軸の最小値を設定します。対数軸の一般的な最小値を使用するには、1 を設定します。ゼロ未満の値を使用するには、使用するデータに適した小さい正の値（1.0E-20 など）を設定します。目盛りラベルに特別なフォーマットを使用するには、***LogLabelsType*** を設定します。

7.1.5 軸の値と画面座標の間の変換

軸には、軸値 (データポイント値) を画面座標に変換し、画面座標を軸値に変換するメソッドがあります。***ValueToCoord*** メソッドを使用して軸値を画面座標に変換し、***CoordToValue*** を使用して画面座標を軸値に変換します。ピクセルが優先される場合は、密度非依存ピクセル(DIP)ではなく、**UseDIP = False** に設定します。

```
float screenCoordinate = _chart.ViewXY.XAxes[0].ValueToCoord(axisValue);
```

ValueToCoord および ***CoordToValue*** メソッドは、チャートの最終サイズが取得された後に使用可能になります。たとえば、***chart.AfterRendering*** イベントにサブスクライブして、チャートが完全にレンダリングされたことを確認します。

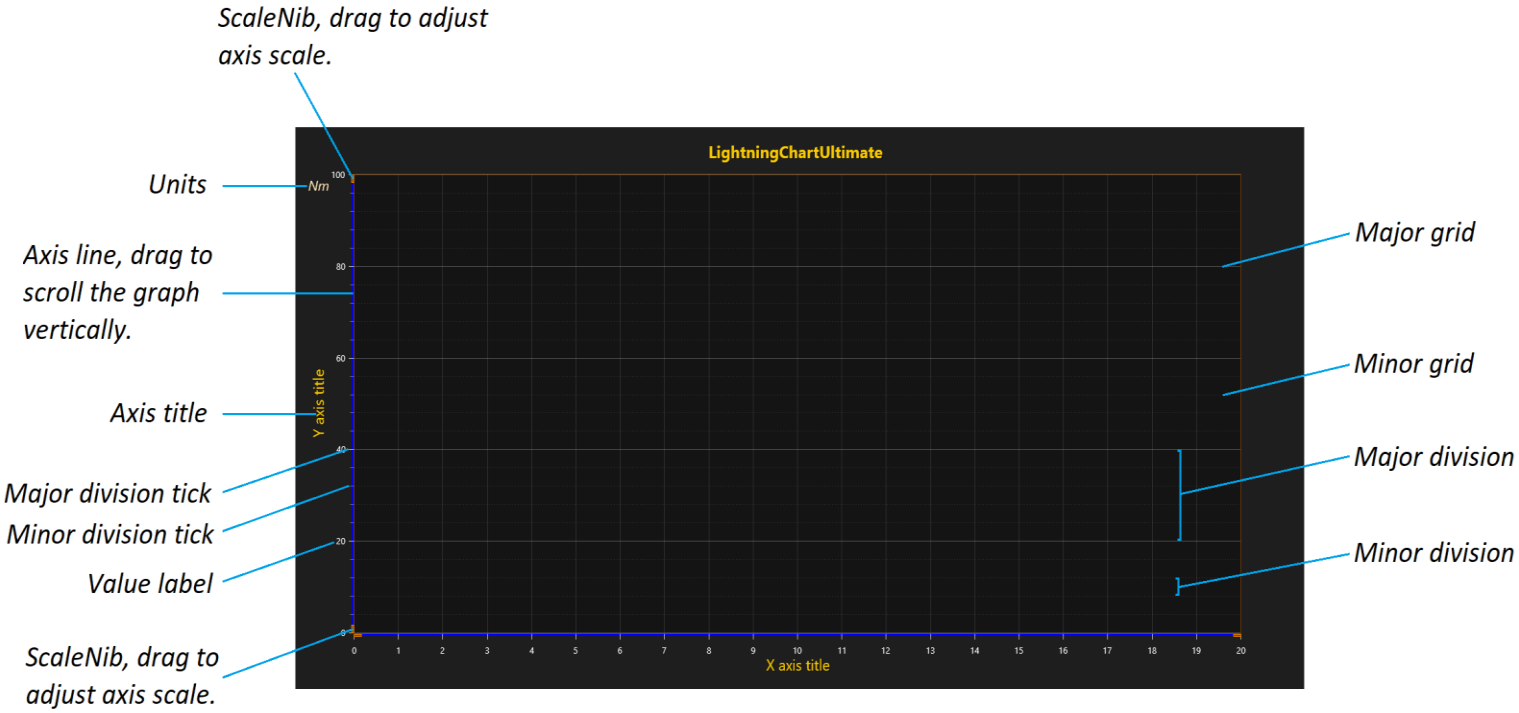
複数の値または座標を一度に変換するには、***ValuesToCoords*** および ***CoordsToValues*** メソッドを使用します。これらは、軸値を double 配列 (X 軸 ***CoordToValue*** の整数配列) として受け取り、画面座標を float 配列として返します。

7.2 Y 軸

Y 軸は無制限の数を定義できます。***YAxes*** コレクションプロパティを使用して、Y 軸を追加します。

```
// Adding Y-axes to the chart
AxisY axisY = new AxisY(_chart.ViewXY);
axisY.Title.Text = "Y-axis";
chart.ViewXY.YAxes.Add(axisY);
```

7.3 X 軸



X 軸の分割とグリッド設定は、Y 軸の設定と同じです。したがって、前の章で説明したすべてのプロパティと機能は、X 軸にも適用できます。ただし、X 軸には、Y 軸にはないリアルタイムスクロール関連のプロパティがいくつかあります。

7.3.1 リアルタイムモニタリングスクロール

リアルタイムモニタリングソリューションを作成する場合、X 軸をスクロールして現在の監視位置を正しく表示する必要があります。これは通常、最新の信号ポイントのタイムスタンプです。新しい信号ポイントがシリーズに設定された後、最新のタイムスタンプを ***ScrollPosition*** プロパティに設定します。

LightningChart には複数のスクロールモードがあります。***ScrollMode*** プロパティを選択して使用します。

注意

デフォルトオプションです。***ScrollPosition*** を ***None*** に設定するとスクロールは適用されません。多くの場合、これはリアルタイムモニタリングを使用しない場合に使用します。

ステッピング

収集されたデータが X 軸の終わりに達すると、すべての系列データを含む軸が、ステップ間隔分だけ左にシフトされます。このシフトは、X 軸の終点に到達するたびに実行されます。***SteppingInterval*** プロパティは、値の範囲として定義されます。

スクローリング

X 軸は、スクロールギャップに達するまで固定され、その後すべてのシリーズの X 軸が連続的に左にシフトされます。スクロール位置が X 軸の最後に達したときにスクロールが有効になる場合は、***ScrollingGap*** を 0 に設定します。***ScrollingGap*** プロパティはグラフ幅のパーセントとして定義されます。

LightningChart は、***series.AddPoints()*** 、***AddValues()*** 、または ***AddSamples()*** メソッドを使用する場 合、リアルタイム信号のインクリメンタルレンダリングデータ構築をサポートします。つまりレンダリングデータは、データの新しい部分からのみ計算され、既存のレンダリングデータと結合されます。

インクリメンタルレンダリングデータ構築を使用するには、次のように新しいポイントを追加します。

```
chart.BeginUpdate();
series.AddPoints(array,false);
xAxis.ScrollPosition = latestXValue;
chart.EndUpdate();
```

一連の ***InvalidateData()*** 呼び出しを使用して、レンダリングデータの完全なリフレッシュをいつでも行うことができます。

スイーピング

スイープは、おそらく最もユーザーフレンドリーなリアルタイムモニタリングビューを提供します。スイープは 2 つの X 軸を使用します。最初の軸が完全に収集された後、スイープギャップが表示されます。次に、2 番目の X 軸が最初の X 軸の上にスイープされます。両方の X 軸には、独自の値ラベルが表示されます。***SweepingGap*** プロパティは、グラフ幅のパーセントとして定義されます。

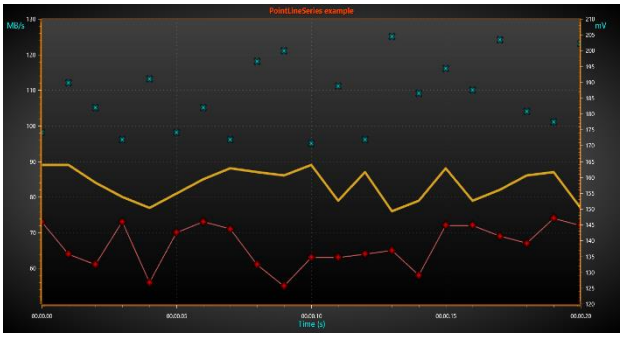
トリガー

X 軸の位置は、トリガーレベルを超過または下回るシリーズ値によって決定されます。***Triggering*** プロパティを使用して、トリガーオプションを設定します。***Triggering.TriggeringActive*** プロパティを有効にするとトリガーをアクティブに設定できます。

1 つのシリーズをトリガーシリーズとして設定する必要があります。許容されるトリガーシリーズタイプは、***PointLineSeries*** および ***SampleDataSeries*** です。***Triggering.TriggerLevel*** でト

リガーY レベルを設定します。***Triggering.TriggeringXPosition*** を使用して、レベルトリガーポイントがグラフ幅のパーセントとして水平に描画される場所を指示します。

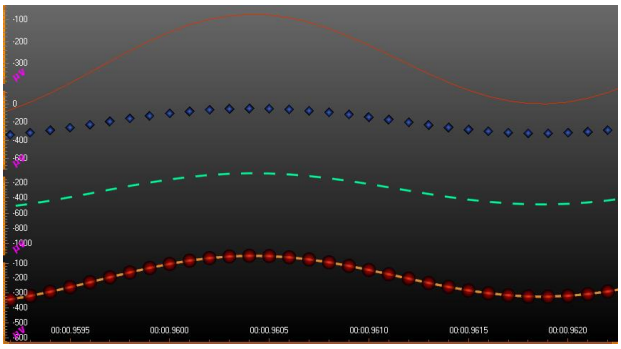
7.4 PointLineSeries



PointLineSeries は、シンプルライン、ポイント（散布）、またはその両方をポイントラインとして表示できます。***PointLineSeries*** オブジェクトを ***PointLineSeries*** リストに追加して、系列をグラフに追加します。

LineStyle プロパティでラインスタイルを定義します。ラインを表示しない場合は、***LineVisible = false*** を設定します。ポイントを表示するには、***PointsVisible = true*** を設定します。***PointStyle*** プロパティを設定して、ポイントスタイルを変更します。***PointStyle.Shape*** にある予め定義された、***Circle***、 ***Triangle***、 ***Cross***、***bitmap*** などのスタイルから、形状を選択します。***PointLineSeries*** も個々のポイントの色付けのサポートをします。

7.5 SampleDataSeries

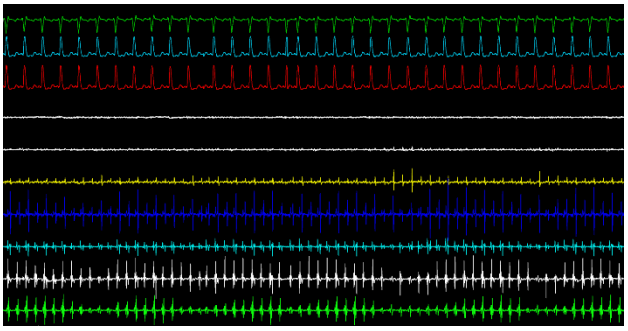


SampleDataSeries は、サンプリングされた信号データ（離散信号データ）を表示するために使用されるラインシリーズです。これは通常、リアルタイム DSP アプリケーションで使用されます。視覚的には ***PointLineSeries*** に似ているため、すべてのラインとポイントの書式設定オプションが適用されます。***SampleDataSeries*** のサンプル間隔は固定されているため、ポイント X の値を保存するためにメモリを予約する必要はありません。

SampleDataSeries *オブジェクトを* ***SampleDataSeries*** リストに追加して、系列をグラフに追加します。***SampleDataSeries*** は、単精度および倍精度のサンプル Y 値をサポートしています。メモリ予約を可能な限り低く保つ場合は、単精度値を使用することを推奨します。***SampleFormat*** プロパティでサンプル形式を選択します。シリーズ ***SamplingFrequency***（1 / サンプル間隔）を使用して、固定サンプル間隔を設定します。サンプルが始まる X 値（タイムスタンプ）を設定するには、***FirstSampleTimeStamp*** プロパティを設定します。

サンプルはコードで追加する必要があります。 ***AddSamples*** メソッドを使用して、既存のサンプルの最後にサンプルを追加します。

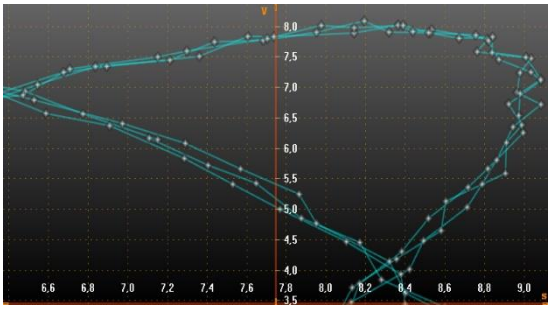
7.6 SampleDataBlockSeries



SampleDataBlockSeries は SampleDataSeries の 1 つのバージョンで、リアルタイムアプリケーションのために完全にオプティマイズ化されています。これは、非常に多くのデータポイントを同時にレンダリングし、CPU とメモリの消費を最小に抑えて可能な限りのベストなパフォーマンスを提供します。そのシリーズ名が示す通り、データはブロックとして内部管理され、それぞれ順番にメモリ管理がされます。これにより、非常に大きな継続リニアメモリを使う必要はありません。***SampleDataBlockSeries*** は、ECG/EKG、EEG などのインダストリアルモニタリングアプリケーション、テレメトリー、ウェーブフォームバイブレーションモニタリングなどのリアルタイムメディカルモニタリングアプリケーションのためのオプティマルシリーズです。

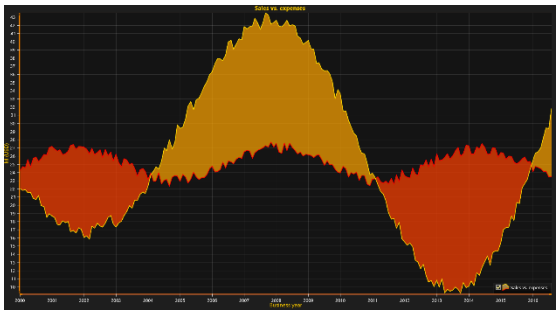
SampleDateBlockSeries は、***SampleDataSeries*** とほぼ同じ働きをします。これは進行順で、固定されたデータ間隔を持っている追加データを必要とします。***SamplingFrequency***（1 /サンプル間隔）を使用して、固定サンプル間隔を設定します。サンプルが始まる X 値（タイムスタンプ）を設定するには、***FirstSampleTimeStamp*** プロパティを設定します。しかしながら、他のラインシリーズに比べ ***SampleDataBlockSeries*** は、視覚的により少ないフォーマットオプションとなります。**色**と**幅**のプロパティは、それぞれラインの色と幅を変更するために使用できます。さらに、***SampleDataBlockSeries*** はラインのみを表示し、個々のポイントは表示しません。

7.7 FreeformPointLineSeries



FreeformPointLineSeries は、シンプルライン、ポイント（散布）、またはその両方をポイントラインとして表示できます。***FreeformPointLineSerie*** では、前のポイントから任意の方向にラインポイントを描画できます。言い換えれば、データ間隔が色々な役目をしている時は、ポイントは進行順である必要はありません。これにより、他のラインシリーズに比べ、***FreeformPointLineSeries*** のレンダリングがやや重くなります。***PokntLineSeries*** からのすべてのラインおよびポイントのフォーマットオプションが適用します。***FreeformPointLineSeries*** オブジェクトを ***FreeformPointLineSeries*** リストへ追加して、シリーズをグラフへ追加します。

7.8 High-lowSeries

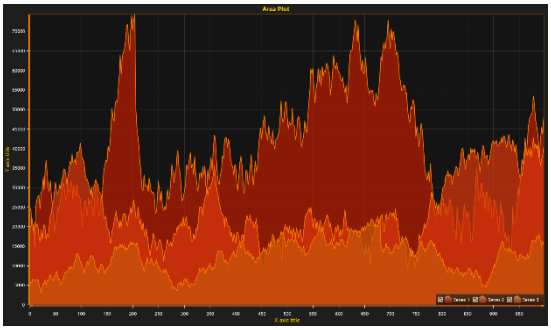


ハイローシリーズは、データを高値と安値の間の塗りつぶされた領域として表します。***HighLowSeries*** オブジェクトを ***HighLowSeries*** リストに追加して、シリーズをチャートに追加します。

塗りつぶしは、***Fill*** プロパティとそのサブプロパティで設定できます。データの高値が安値よりも小さい場合、その部分に ***ReverseFill*** が適用されます。ラインとポイントのスタイルについては、別にハイとローライン用のプロパティがあります(***LineStyleHigh*** および ***LineStyleLow***)。

UseLimitsを有効にすると、シリーズは超過限度を超えた、あるいは下回るさまざまな単色を表示します。通常の***Fill***と***ReverseFill***は、制限間の範囲にのみ適用されます。***UsePalette***を有効にすると、塗りつぶしは***ValueRangePalette***ステップを使用します。**均一**と**グラデーション**の両方の色がサポートされています。データ値はコードに追加する必要があります。データは、**Points[i+1].X ≥ Points[i].X** の X 値の昇順で指定する必要があります。***AddValues(HighLowSeriesPoint[], bool invalidate)*** メソッドを使用して、既存の値配列の最後にデータ値を追加します。

7.9 AreaSeries

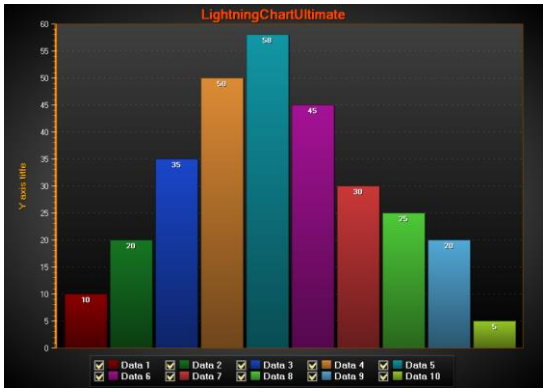


エリアシリーズはベースレベルと値の間の塗りつぶされたエリアとしてデータを表します。エリアシリーズは、***HighLowSeries*** と非常に似ていますが、よりシンプルです。 ***AreaSeries*** オブジェクトを ***AreaSeries*** リストに追加して、系列をグラフに追加します。

BaseValue プロパティでベースレベルを設定します。***Fill*** プロパティを使用して、優先する塗りつぶしスタイルを設定します。ラインのスタイルは ***LineStyle*** プロパティで、ポイントのスタイルは ***PointStyle*** プロパティでそれぞれ設定できます。***HighLowSeries*** のように、超過を制限できます。

データ値はコードに追加する必要があります。データは、**Points[i+1].X ≥ Points[i].X** の X 値の昇順で指定する必要があります。***AddValues(AreaSeriesPoint[], bool invalidate)*** メソッドを使用して、既存の値の配列の最後にデータ値を追加します。古いデータを上書きしながらシリーズデータ全体を一度に設定するには、新しいデータ配列を直接割り当てます。

7.10 BarSeries



BarSeries は水平バーまたは垂直バーにデータを表示できます。***Values*** の配列プロパティを使用して、バーシリーズの値を保存します。 ***AddValue(…)*** メソッドで値を追加します。***SetValue(…)***メソッドを使用して指定された値インデックスで既存の値を更新します。値のタイプは ***BarSeriesValue*** で、次のフィールドがあります。

- ***Value*** バーの長さ
- ***Location*** バーの X 軸位置 (垂直表示) または、Y 軸位置 (水平表示)
- ***Text*** バーに表示されるテキスト

チャートの ***BarViewOptions*** プロパティを使用して、バーの表示方法を設定します。***BarView.Options.Orientation*** で、バーの ***Horizontal*** と ***Vertical*** を選択できます。***BarViewOptions.Grouping*** では、値のインデックス、幅のフィッティングを使用したインデックス、または位置の値によって、バーをグループ化できます。さまざまなバーシリーズの値を視覚的にまとめます。グループ化が不要な場合は、***BarViewOptions.Grouping.ByLocation*** を使用して、***BarSeriesValue*** オブジェクトごとに異なる ***Location*** フィールドを設定します。

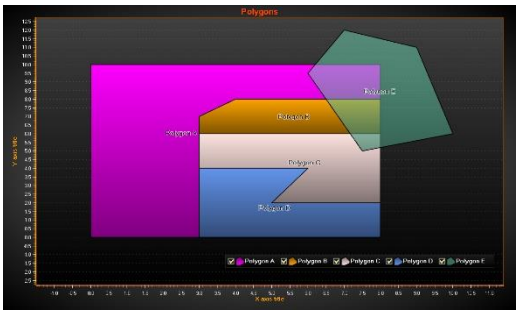
7.11 StockSeries

ストックシリーズを使用すると、ローソク足またはストックバー形式で証券取引所データを視覚化できます。***StockSeries*** リストプロパティに複数の ***StockSeries*** オブジェクトを追加することにより、同じチャートに複数のストックシリーズを追加できます。***Style*** プロパティでスタイルを選択します。オプションは、***Bars***、***CandleStick***、および ***OptimizedCandleStick*** です。

ColorStickDown、***ColorStickUp***、***FillDown***、***FillUp*** プロパティを使用して、色と塗りつぶしのオプションを設定します。***StickWidth*** プロパティでスティックの幅をピクセル単位で調整します。***ItemWidth*** プロパティを使用して、データ項目の合計幅を調整します。***StockSeries*** は、***Behind = True*** を設定することでラインシリーズの前にレンダリングするように設定できます。

データを ***StockSeries*** に設定するために、データ配列を作成し、配列アイテムを設定します。各アイテムには、以下のフィールドがあります。***Date***（年月日）、***Open*** *(*その日の始値) 、***Close*** (その日の終値) 、***Low*** *(その日の最低値)* 、***High*** (その日の最高値) 、***Transacxtion*** (総取引額、オプション) 、***Volume*** (取引された株式数、オプション) 。データ値の昇順でデータを保管してください (古いデータが先) 。

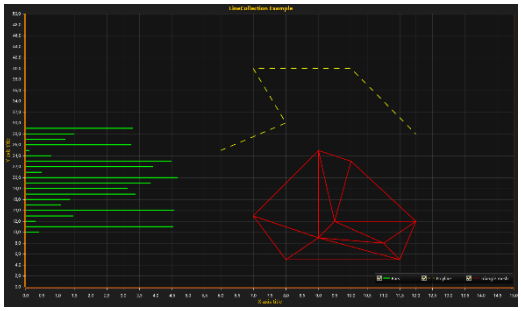
7.12 PolygonSeries



PolygonSeries は、指定された境界線パスで塗りつぶしと境界線をレンダリングします。

Fill プロパティで塗りつぶしを設定します。***PolygonSeries*** の ***Border*** プロパティを使用して境界線のスタイルを設定します。***Points*** プロパティでパスポイントを設定します。 ***PolygonSeries*** には自動パスクローズ機能があります。最後のポイントが最初のポイントに接続されていない場合 、チャートがそれを自動的に行います。

7.13 LineCollections



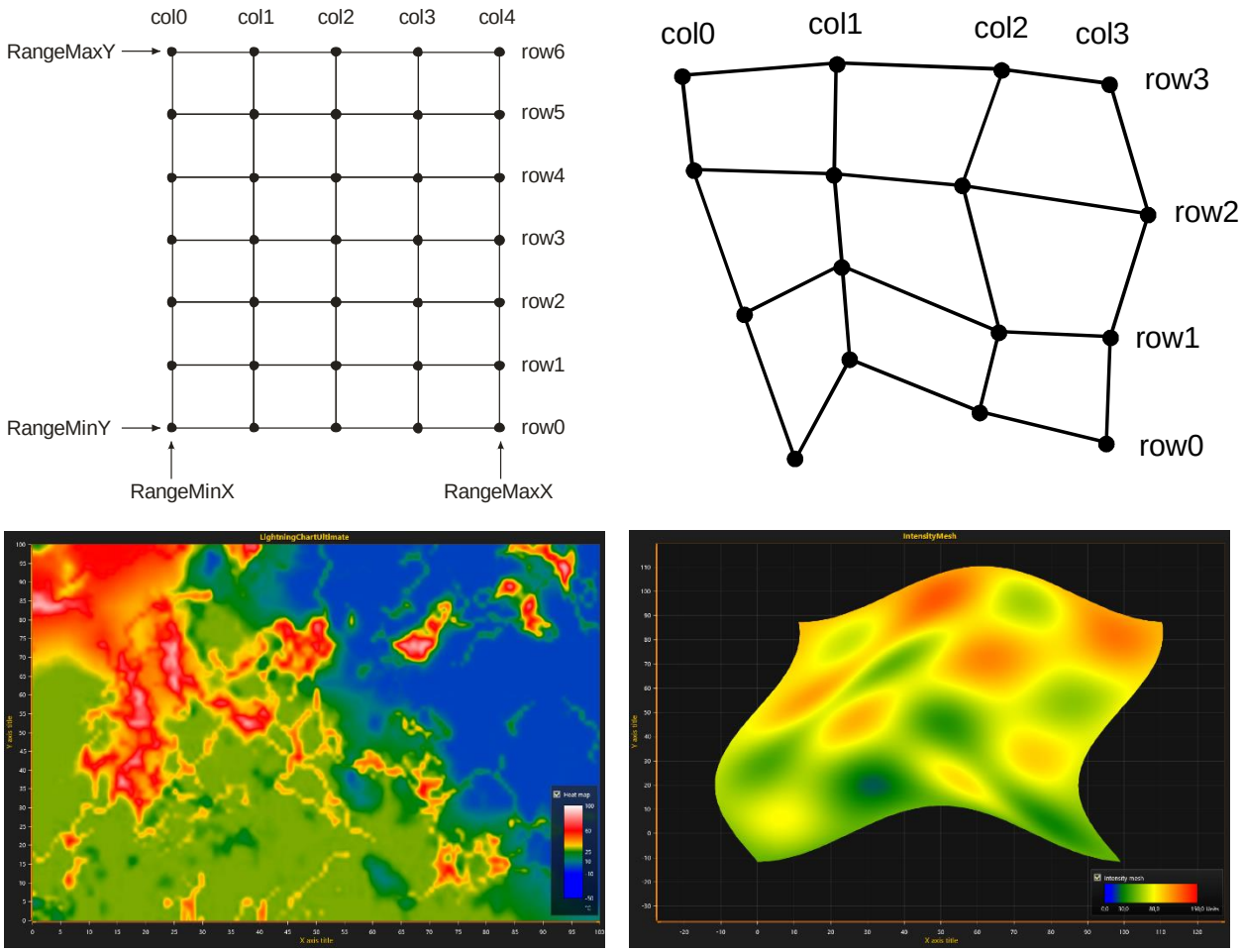
LineCollection は、ラインセグメントのコレクションです。各ラインセグメントは、ポイント A から B に向かうラインで、4 つのフィールドを含んでいます (開始点 X および Y、終点ト X および Y) 。1 つの ***LineCollection*** には、数千のラインセグメントを含めることができます。***LineCollection*** は、***PointLineSeries***、***FreeformPointLineSeries*** または ***SampleDataSeries*** とは対照的に、数千の異なるラインセグメントのレンダリングにおいて非常に効率的です。***PointLineSeries***、***FreeformPointLineSeries*** または ***SampleDataSeries*** は、数百万のポイントの連続したポリラインのレンダリングにおいてより効率的です。

ViewXY.LineCollections リストプロパティに ***LineCollection*** オブジェクトを追加します。ラインカラー、スタイル、幅をコントロールするためには、***LineStyle*** プロパティを使います。***Lines*** プロパティでラインセグメントを設定します。***SegmentLines*** 配列を ***Lines*** プロパティへ下記の通り追加します。

```
lineCollection.Lines = new SegmentLine[] {
        new SegmentLine(6,25,8,30),
        new SegmentLine(10,40,12,28) };
```

7.14 IntensityGrid- および IntensityMeshSeries

IntensityGridSeries および***IntensitveMeshSeries***では、割り当てられた値の範囲のパレットで色分けされたノードのM x N配列を視覚化できます。ノード間の色は補間されます。***IntensityGridSeries*** は、XおよびY次元の等間隔の長方形シリーズです。一方、***InternsityMeshSeries***においてノードは, X-Yスペースに任意で配置する事ができます。両シリーズともに、等高線、等高線ラベル、ワイヤフレームがレンダリングできます。



左図 IntensityGridSeries、右図 IntensityMeshSeries

Fill プロパティを使用して塗りつぶしスタイルを選択します。***Paletted*** の塗りつぶしを使用する時は、***VelueRangePalette*** 使い、値の色付けの色ステップを定義します。パレットは、***MiniValue***、***Type*** および ***Steps*** プロパティで定義されます。***Type*** には、***Uniform*** と ***Gradient*** の 2 つの選択肢があります。***ValueRangePalette*** は、***Fill***、***Wireframe***、***Contour lines*** に使う事ができます。等高線パレットのために数個のステップを定義します。各ステップには、高さの値と相当する色があります。

注意! 20 ステップまでは、プリコンパイルされており高速でロードされます。これ以上ステップ数が多い場合、チャートの初期化に数秒の遅延が予想されます。

データは***Data***プロパティに2次元配列として保存されます。各配列項目は、***IntensityPoint***タイプです。***IntensityPoint***構造の***Value***フィールドに各ノードのデータ値を保存します。これは、***ValueRangePalette***から使用する色を指示します。IntensityMeshのジオメトリ、またはIntensityGridシリーズのSizeXまたはSizeYが変化しない場合、グリッドをアップデートするには、***SetValuesData***あるいは、***SetColorsData***メソッドを使う事が最も有利です。グリッド/メッシュをリフレッシュするために***InvalidateData()*** メソッドを呼び起こします。

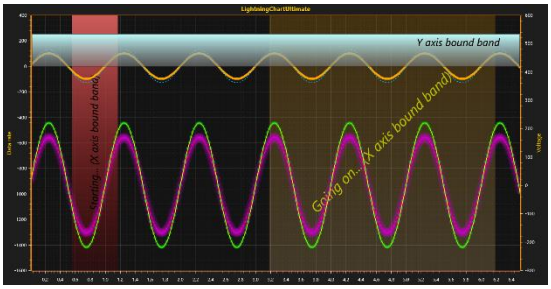
強度グリッドデータの設定

- ***RangeMinX*** プロパティと ***RangeMaxX*** プロパティを使用して X 範囲を設定し、割り当てられた X 軸の最小値と最大値を並べます。
- ***RangeMinY*** プロパティと ***RangeMaxY*** プロパティを使用して Y 範囲を設定し、割り当てられた Y 軸の最小値と最大値を並べます。
- ***SizeX*** および ***SizeY*** プロパティを設定して、グリッドに列と行のサイズを与えます。
- 各ノードの**値**を設定します。

強度メッシュデータの設定

- Set ***SizeX*** および ***SizeY*** プロパティを設定して、メッシュに列と行のサイズを与えます。
- すべてのノードのために X, Y および Value を設定します。
- ジオメトリが変わらない場合は、 ***Optimization*** to ***DynamicValuesData*** に対して ***Optimization*** を設定します。
- ジオメトリが変わらない場合は、すべてのノードに値を設定するだけでデータを更新します。ジオメトリが変わる場合は、すべてのノードに対し、X,Y および valume を更新します。

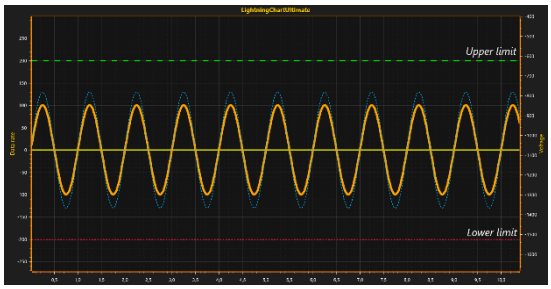
7.15 バンド



バンドは、マージンから別のマージンに達する垂直または水平の領域です。***Binding*** プロパティを使用して、バンドを Y 軸または X 軸にバインドできます。バンドが Y 軸にバインドされている場合、***AssignYAxisIndex*** プロパティも設定する必要があります。

バンドエッジは、バインドされた軸の値である ***ValueBegin*** および ***ValueEnd*** プロパティによって設定されます。バンドはマウスで別の場所にドラッグできます。バンドをエッジからドラッグしてサイズを変更すると、ドラッグされたエッジの ***ValueBegin*** または ***ValueEnd*** が更新されます。

7.16 定数線



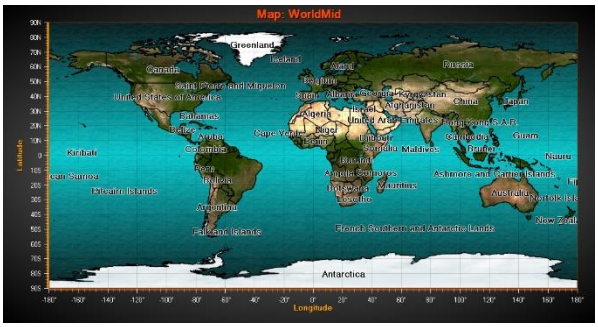
バンドと同様に、定数線はシリーズと見なすことができます。定数線は Y 軸にバインドされ、グラフの左端から右端までの 1 本の水平線を表します。***Value*** プロパティでレベルを設定します。

定数線は、マウスでドラッグすることにより垂直に移動できます。***Behind*** プロパティを true に設定すると、定数線がラインおよびバーシリーズの背後に描画されます。

7.17 地図

Maps プロパティとそのサブプロパティを使用して、地理的な地図を表示します。LightningChart マップには、***vector maps*** と ***tile maps*** の 2 つの異なるカテゴリがあります。マスは、いわゆる*正距円筒図法*で表示されます。この投影により、LightningChart のシリーズタイプと、実質的にすべてが X 軸と Y 軸にバインドされている他のオブジェクトを、地図と同時に使用できます。

7.17.1 ベクトルマップ



地理的ベクトルデータは、***.md*** 拡張子の LightningChart マップファイルに保存されます。 LightningChart には、一連のマップファイルが付属しています。

X 軸は経度に、Y 軸は緯度に使用されます。地図の座標は 10 進度で、緯度の原点は赤道、経度の原点は英国グリニッジです。

マップファイルが存在する ***Path*** プロパティにディレクトリ名を設定します。アクティブなマップは、LightningChart で提供されるマップの ***Type*** プロパティで選択できます。独自のマップファイルを使用するには、***FileName*** プロパティを設定します。マップが必要ない場合は、***Type*** を ***Off*** に設定します。
各マップファイルには複数のレイヤーを含めることができます。例えば、陸地、湖、川、道路、都市のレイヤーがあります。レイヤーとそのデータは、***Layer*** 配列プロパティからアクセスできます。レイヤーには特定のタイプがあります。レイヤーの外観オプションは、対応するオプションプロパティで変更できます。例えば、***LandOptions*** を使用して土地地域の外観を変更します。

マウスの双方向性

マップ領域およびオブジェクトとのあらゆる種類の相互運用に対して ***AllowUserInteraction*** を有効にします。領域（土地、湖）とベクターレイヤー（川、道路）をマウスでポイントできます。マウスがオブジェクト上にあると、オブジェクトは、***Highlight*** プロパティに従ってハイライトされます。***Hightlight*** を ***None*** に設定すると、オブジェクトは強調表示されませんが、クリックすると ***Maps.ButtonDownOnMapItem*** イベントの呼び出しなどに使用できます。

マップオブジェクトには、人口やその他の統計データなどの関連データが含まれている場合があります。 ***UserInteractiveDeviceOverOnMapItem/UserInteractiveDeviceOverOffMapItem/ButtonDownOnMapItem*** イベントハンドラーを使用してデータにアクセスします。マップアイテムのデータは ***GetInfo*** メソッドで取得でき、キーと値の辞書を提供します。

ESRI 形状ファイルデータからマップをインポートする

インポート機能により、.shp ファイルから LightningChart マップファイル（.md）が作成されます。 ESRI シェイプファイル（*.shp）は、ベクターおよびポリゴンデータをサポートする広く使用されているマップファイル形式です。

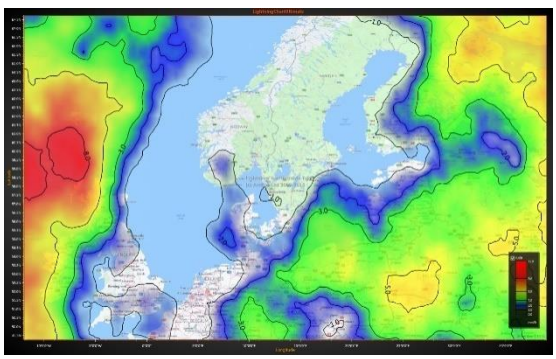
マップウィザードを使用して、シェープファイルデータを LightningChart（LC）マップデータ形式に変換できます。 LC 形式は階層化をサポートしているため、複数のシェープファイルを 1 つのファイルにマージできます。マップファイルの構造とオブジェクトは、実行時のパフォーマンスを最大にするために前処理されます。

アドバイス *LightningChart®.NET のデモアプリケーションには、マップのインポートの例があります。そこからインポートウィザードを実行して、インポートを通じてカスタム LC マップファイルを作成します。*
変換は少なくとも 3 つのステップで実行されます。

1. Shapefile Selection Dialog のファイルに基づいてファイルを選択し、レイヤーに設定します。
2. ファイルテキストエンコーディングの決定
3. 結果のマップファイルに含まれるアイテムを選択します。

元の shp ファイルごとにステップ 2 と 3 が繰り返されることに注意してください。シェープファイルは使用するエンコーディングを認識しないため、ユーザーが選択する必要があります。

7.17.2 タイルマップ



LightningChart は現在、Here マップをサポートしています。 (ストリートマップと衛生画像。Here サーバーを使用するために、開発者あるいはエンドユーザーは、Here と契約をする必要があります。)

ViewXY.Maps.TileLayers コレクションに ***TileLayer*** オブジェクトを追加します。***AlphaLevel*** プロパティを

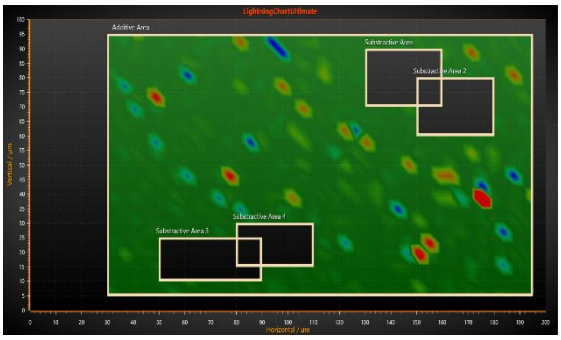
使用して、複数のレイヤーを挿入して半透明にすることができます。***TileLayer*** オブジェクトは、***TileLayers*** コレクションの外観の順にレンダリングされます。最初のレイヤーは背景にあります。***AboveVectorMap = False*** を設定することにより、定義されている場合はレイヤーがベクトルマップの前にレンダリングされます（第 **2.1.1** 章を参照）。デフォルトでは、***TileLayer*** はベクターマップの後にレンダリングされます。

TileLayer は、http プロトコルを介してオンラインサービスプロバイダーから小さな画像として情報を取得し、チャートエリアに表示します。マップビューをズームまたはパンすると、画像が更新されます。

チャートはタイルをキャッシュフォルダーに格納します。これにより、同じ領域で頻繁にパンまたはズームする際の読み込み時間が大幅に短縮されます。チャートでタイルを表示する必要がある場合、まずキャッシュフォルダーにあるかどうかを確認し、見つからない場合は Web サービスから取得します。デフォルトでは、キャッシュフォルダーは ***c:\Users\[Current user]\AppData\Local\Temp*** です。

ViewXY.Maps.TileCacheFolder にキャッシュフォルダーを設定します。
ViewXY.Maps.ClearTileCacheFolder() メソッドを呼び出して、キャッシュフォルダーをクリアします。

7.18 StencilAreas

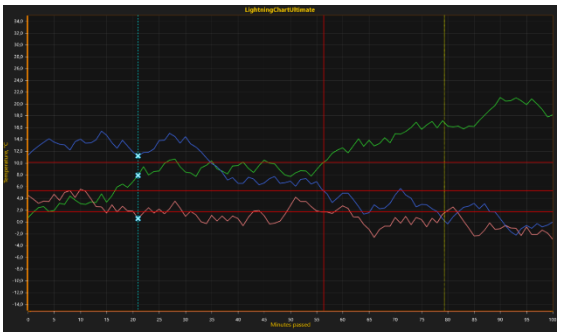


IntensityGridSeries、***IntensityMeshSeries***、および ***Maps*** には、描画データの領域内または領域外をマスクできる ***StencilArea*** 機能があります。***StencilArea*** は、新しい ***StencilArea*** を作成して適用する事ができます。そして、そのサイズを ***AddPolygon()*** を介して PointDouble2D-array として、あるいは、***AddMapLayerIndex()*** を介してマップレイヤーとして 定義し、最終的には、それらをマスクされるシリーズに追加します。複数の ***StencilAreas*** を設定する事が可能です。2 つ、あるいは、それ以上のエリアが重なった場合は、エリアは接合されます。

- ***AdditiveAreas*** は正のステンシルマスクを作成します。領域内のデータのみが描画され、外側はクリップされます。
- ***SubtractiveAreas*** はネガティブステンシルマスクを作成します。エリア内のデータはクリップされ、外側が描画されます。***SubtractiveAreas*** は、***AdditiveAreas*** とのみ連携するように設計されていることに注意してください。これらがないと、クリッピングは適用されません。

StencilArea オブジェクトがリスト（***AdditiveAreas*** または ***SubtractiveAreas***）に追加されるたびに、それぞれのシリーズに対して ***InvalidateStencil()*** または ***InvalidateData()*** を呼び出す必要があります。またステンシルを定義するポイントの配列を時計回りに設定することを推奨します。

7.19 LineSeriesCursors



ラインシリーズカーソルを使用すると、X 座標で値を追跡することによりラインシリーズデータを視覚的に分析できます。シリーズ値は、***ITrackable*** インターフェイスを実装するシリーズ（***SampleDataSerie***、***PointLineSeries***、***AreaSeries***、***HighLowSeries***）でのみ解決できます。他のシリーズタイプの場合、Y 座標はカーソルによって自動的に追跡されません。

LineSeriesCursors コレクションに ***LineSeriesCursor*** オブジェクトを追加します。***SnapToPoints*** を有効にして、カーソルをポイントからポイントにジャンプします。***Style*** プロパティでカーソル追跡スタイルを設定します。 ***Style*** が ***PointTracking*** に設定されている場合、ビットマップイメージであっても、任意の追跡ポイントスタイルを使用できます。***HairCrossTracking*** スタイルを使用する場合、ラインシリーズポイント Y 値に水平線が描画されます。同じシリーズの複数のポイントがカーソル位置でヒットした場合、最小ポイントと最大ポイントの中央に線が引かれます。

LineSeriesCursor の位置でデータ値を解決する

ITrackable インターフェイスを実装するシリーズは、X スクリーン座標または X 軸値によって解決できます。正確な方法の ***SolveYValueAtXValue*** は、必要に応じてデータポイントをループし、最も近いデータポイントを見つけます。粗い方法の ***SolveYCoordAtXCoord*** はシリーズのキャッシュされたレンダリングデータを使用して、一致する Y スクリーン座標を解決します。***AnnotationXY*** オブジェクトが、カーソルの隣の値を表示するためによく使われます。

7.20 EventMarkers

EventMarkers を使用すると、リアルタイムモニタリング中に何か特別なことが発生した場合、またはデータに特別な注釈を付けたい場合に、関心のあるポイントをマークできます。***Symbol*** プロパティでマーカーシンボルを定義し、***Label*** プロ

パティでテキストラベルを定義します。 ***VerticalPosition*** プロパティで垂直位置を設定し、必要に応じて ***Offset*** を使用してオブジェクトプロパティをシフトします。すべてのマーカーに ***XValue*** を割り当てることで X 軸上のマーカーの位置が設定されます。マーカーの形を選択するために、***Symbol.Shape*** を設定します。

チャートイベントマーカー

シリーズイベントマーカーとは異なり、***ChartEventMarkers*** *は*、特定のシリーズに添付されません。マーカーはマウスで別の場所にドラッグできます。

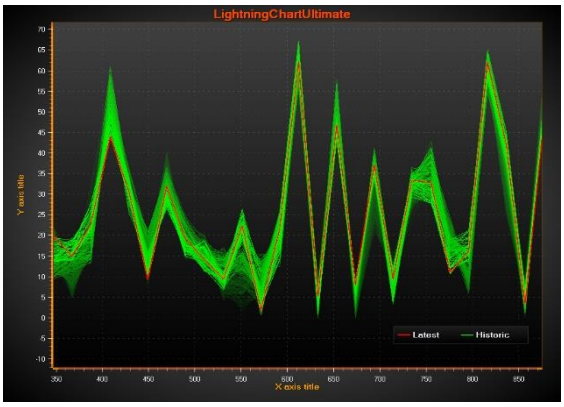
チャートマーカーの位置は、***VeritcalPosition, XValue***、 および ***Offset*** プロパティを介して設定できます。さらに、***BindToXAxis*** を true に設定すると、マーカーが特定の X 軸にバインドされます。実際はこれによりマーカーは現在の X 値にとどまり、X 軸がパンされたときなどに移動します。***BindToXAxis*** を無効にすると、軸の移動方法に関係なくマーカーは同じチャート位置に保持されます。

ラインシリーズイベントマーカー

系列シリーズには ***SeriesEventMarkers*** コレクションプロパティがあります。シリーズ固有のイベントマーカーを割り当てるために使用できます。シリーズイベントマーカーは、イベントマーカーをシリーズ値に添付したままマウスで別の場所にドラッグできます。これを有効にするには、マーカーの ***VerticalPosition*** を ***TrackSeries*** に設定する必要があります。これは、***ITrackable*** インターフェイスを実装するシリーズで利用可能です。

HorizontalPosition を ***SnapToPoints*** に設定することにより、マーカーは最も近いデータポイントの位置に水平に整列します。***HorizontalPosition = AtXValue*** ではマーカーを任意の X 値に配置でき、***VerticalPosition = AtYValue*** ではマーカーを任意の Y レベルに垂直に設定できます。

7.21 永続的なシリーズレンダリングレイヤー



PersistentSeriesRenderingLayer は、繰り返しのライン/ポイントデータ、または同じ X および Y 範囲で何度もプロットされるライン/ポイント/ハイロー/面積塗りつぶしデータの非常に高速なレンダリングに使用できます。***PersistentSeriesRenderingLayer*** はビットマップの一種であり、レンダリングデータを段階的に追加できます。コマンドによってクリアされるまでグラフィックを保持します。この方法では、各更新ラウンドで 1 つのシリーズのみをレイヤーにレンダリングする必要があり、その後にレイヤーが画面にレンダリングされ

ます。CPU 負荷またはメモリフットプリントは増加しません。既存のデータを徐々にフェードアウトする必要がある場合は、ビットマップピクセルのアルファを乗算することで実行できます。

必要な数だけ ***PersistentSeriesRenderingLayer*** オブジェクトを作成することが可能であり、各ラウンドで任意の数のシリーズをレンダリングできます。Any of ***PointLineSeries***, ***SampleDataSeries***, ***FreeformPointLineSeries, HighLowSeries*** or ***AreaSeries*** オブジェクトのいずれも、レンダリングしてレイヤーにする事ができます。

レイヤーを作成する

PersistentSeriesRenderingLayer オブジェクトは、コードで作成する必要があります。

```
using Arction.[edition].Charting.Views.ViewXY;

PersistentSeriesRenderingLayer layer = new PersistentSeriesRenderingLayer (m_chart.ViewXY,
m_chart.ViewXY.XAxes[0]);
```

描画されるシリーズで使用されるものと同じ XAxis オブジェクトを提供します。そして、1 つあるいは、複数のシリーズをそのレイヤーでレンダリングするために 、***layer.RenderSeries()*** を使います。

MultiplyAlpha(value) は、レイヤーをより透明または不透明にすることができます。レイヤー内のすべてのピクセルを個別に乗算します。1 未満の値を指定すると、透明度が増加します (レイヤーが減衰します) 。1 より大きい値を指定すると、不透明度が高くなります (レイヤーが見やすくなります) 。

layer.Clear() はレイヤーをクリアし、ARGB = （0,255,255,255） で色を初期化します。***layer.Clear(Color color)*** は指定された色でレイヤーをクリアします。ほとんどの場合、背景で使用されているのと同じ色を設定するのが最も便利ですが、A = 0 に設定されます。レイヤーを破棄し、チャートでレンダリングされないようにするには、***layer.Dispose()***を呼び出します。

知っておくべきレイヤーの制限

特別なレンダリング技術のため、これらの制限に留意してください。

- X 軸の ***ScrollMode*** は ***None*** に設定する必要があります。このアプローチでは、X 軸のリアルタイムスクロールは不可能です。

-ズーム、パン、軸調整、およびチャートのサイズ変更により、画像は軸範囲と同期しなくなります。永続的なプロットを使用する場合はこれらの機能を無効にするか、レイヤーをクリアし、新しいレイヤーレンダリングのために古いラインシリーズを一時的に再作成するようにア

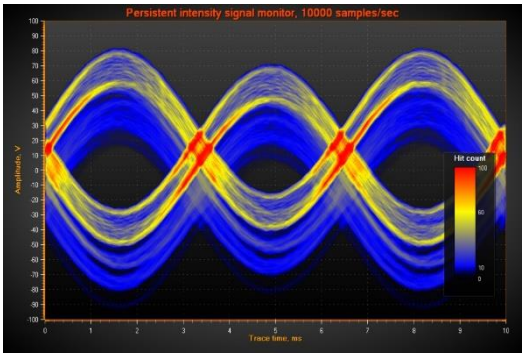
プリケーションロジックを作成する必要があります（軸範囲の変更とサイズ変更のイベントハンドラーがあります）。

-グラフのサイズ変更によりレイヤーがクリアされ、Windows デスクトップのロック状態から再開されます。

-マウスインタラクティビティは、レイヤーでのみレンダリングされるシリーズではサポートされていません

- EMF / WMF / SVG のエクスポート、ベクター形式でのクリップボードへのコピー、ベクター形式での印刷はレイヤーをサポートしていません。ラスタ形式のみがサポートされています。

7.22 永続的なシリーズレンダリング強度レイヤー



PersistentSeriesRenderingIntensityLayer を使用すると、トレースをレイヤーに収集し、ピクセルごとのヒットカウントで色付けできます。色付けは、値の範囲のパレットを使用して行われます。トレースは、***PersistentSeriesRenderingLayer*** と同じシリーズタイプです。2 回目のレンダリング呼び出しでトレースをピクセルの位置で再度レンダリングすると、その強度が大きくなり、値範囲パレットで値が増加します。

レイヤーを作成する

PersistentSeriesRenderingIntensityLayer オブジェクトは、コードで作成する必要があります。

```
PersistentSeriesRenderingIntensityLayer layer = new
PersistentSeriesRenderingIntensityLayerngLayer(m_chart.ViewXY, m_chart.ViewXY.XAxes[0]);
```

レイヤーの ***ValueRangePalette*** プロパティでパレットタイプとステップを定義します。***ValueRangePalette.Type = Gradient*** は、グラデーションの色付けを行い、***ValueRangePalette.Type = Uniform*** は、レイヤーを個別のカラーステップでレンダリングします。 その後、***layer.RenderSeries()*** を使って 、レイヤーに 1 つ、あるいは複数のシリーズをレンダリングします。

layer.Clear() は、レイヤーをクリアし、カウンターをリセットします。レイヤーを破棄し、チャートでレンダリングされないようにするには、 ***layer.Dispose()*** を呼び出します。

新しいトレースの強度効果と古いトレースの減衰の調整

NewTraceIntensity プロパティを使用して、RenderSeries 呼び出しでレンダリングされた新しいトレースが取得する強度を設定します。典型的な値は 1〜100 で、色範囲がトレースでいっぱいになるまでの速さによって異なります。

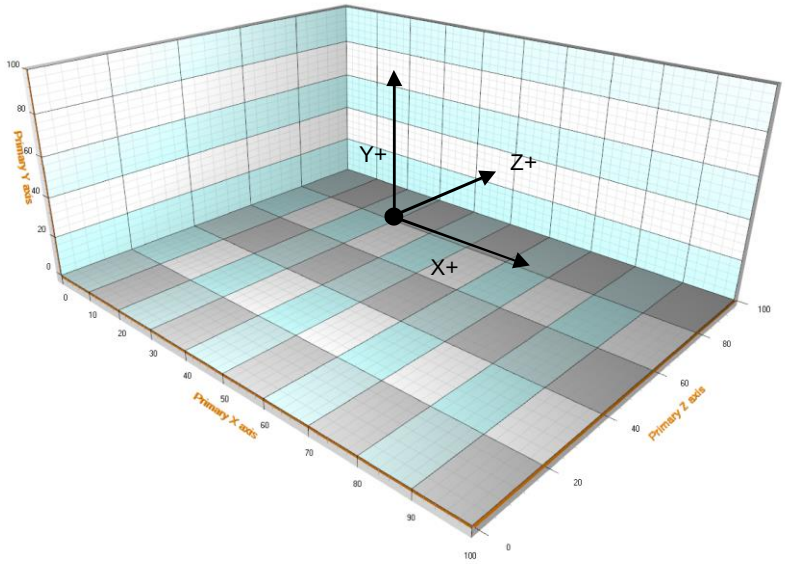
HistoryIntensityFactor を使用して、古いトレースの減衰速度を調整します。典型的な値の範囲は 0.5〜0.99 です。

データがレイヤーに更新されると ***NewTraceIntensity*** が新しいトレースに使用されます。古いトレースデータは ***HistoryIntensityFactor*** で同時に減衰します。すべてのシリーズオブジェクトの後に ***layer.RenderSeries(List<PointLineSeriesBase> seriesList)*** は古いトレースを減衰させます。

8. View3D

View3D では、3D 空間でデータを視覚化できます。 3D モデルは、さまざまな方法でズーム、回転、およびライトアップできます。異なるシリーズタイプを同じ 3D ビューに配置して、組み合わせて視覚化することができます。

3D モデルは 3D の世界の中心に構築されます。寸法の大きさは、3D 空間でのモデルボックスのサイズを定義します。壁と軸のサイズは、この寸法ボックスで定義されます。 ***Dimensions*** プロパティを使用して、各次元の大きさを設定します。
カメラの回転が定義されていない場合、正の X 方向は右、正の Y 方向は上方向、正の Z 方向は画面の内側です。



一部の 3D オブジェクトは、軸の値ではなく「*世界座標*」を使用します。例えば、ライトはこのように配置されて、軸範囲から独立しています。原点[0,0,0] はモデルの中心にあります。実際の 3D モデル空間の範囲は、[-Dimensions.X/2 から Dimensions.X/2]、[-Dimensions.Y/2 から Dimensions.Y/2]、および[-Dimensions.Z/2 から Dimensions.Z/2]になります。

8.1 壁

壁（***WallOnFront***、***WallOnBack*** その他）は軸グリッドとグリッドストリップを表示し、軸のベースを提供するために使用されます。デフォルトでは、下、左、右、背面、および前面の壁が表示されます。***AutoHide*** プロパティは true に設定されています。ビューを回転させると邪魔な壁が一時的に非表示になり、チャートのコンテンツの表示が妨げられなくなります。壁を強制的に表示するには、***Visible = true*** および ***AutoHide = false*** に設定します。

XGridAxis、***YGridAxis***、***ZGridAxis***、***GridStripColorX***、***GridStripColorY***、***GridStripColorZ*** および ***GridStrips*** プロパティを使用してグリッドを適用する軸を選択し、グリッドストリップの色を変更します。利用可能なプロパティは、壁の向きによって異なります。***FullTransparent*** プロパティは、ソリッドな壁が隠れている時に、グリッドだけを表示させます。***FullTransparent*** が有効でも、グリッドは依然として、壁の ***Visible*** および ***AutoHide*** プロパティに従う事に注意してください。

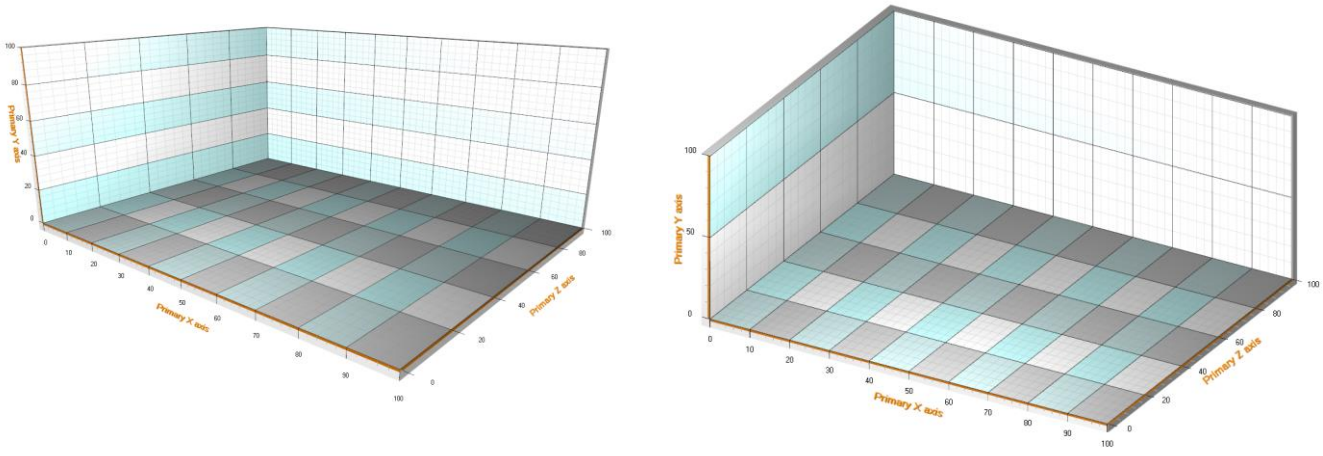
壁の代わりに、簡素化された 3D ボックスプレゼンテーションを使用できます。すべての壁に Visible = false を設定してから、***FrameBox.Style = AllEdges*** を設定します。***FrameBox.LineColor*** で色またはフレームを設定します。

8.2 カメラ

カメラのタイプ、場所、距離、およびターゲットにより 3D ビューポイントが決定されます。***RotationX***、***RotationY***、***RotationZ***、***ViewDistance*** を使用して、3D モデル空間でカメラの位置を設定します。***Target*** プロパティを設定して、カメラを優先方向に向けます。

Projection プロパティで投影タイプを選択します。

- ***Perspective*** は現実的な投影を示しています。
- ***Orthographic*** は科学および工学用途で使用される正射投影タイプです。***OrthographicLegacy*** よりも推奨されます。
- ***OrthographicLegacy*** は、3D ワールド座標（軸値ではない）で定義されている場合、3D オブジェクトのサイズを維持します。これは正射投影に比べて、ズーム後のレンダリングが遅くなります。

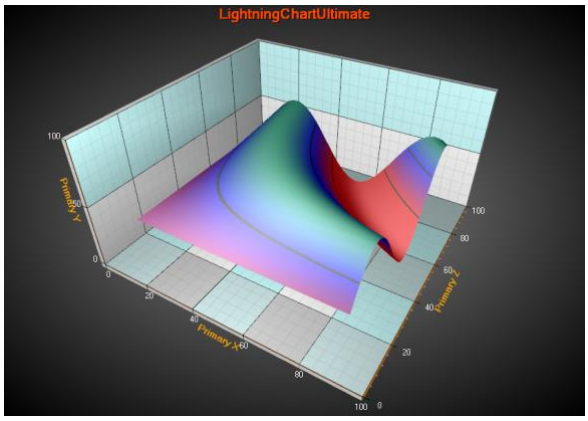


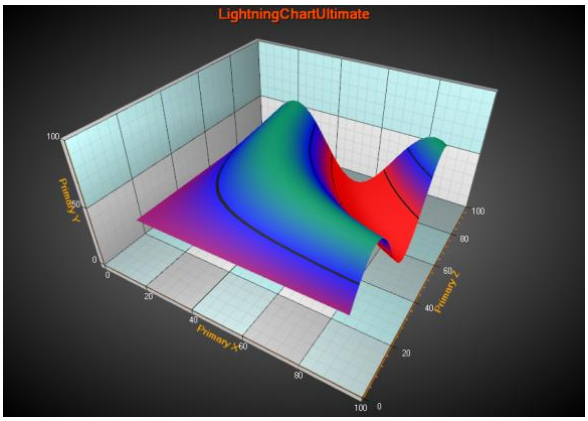
3D 空間の遠近法および正投影かメラビュー

8.3 ライトとマテリアル

ライトは、3D モデル空間のどこにでも自由に配置できます。いくつかのライトを ***Lights*** コレクションプロパティに追加できます。 ***Directional*** と ***PointOfLight*** の 2 種類のライトタイプがあります。

すべての 3D オブジェクトには ***Material*** プロパティがあります。マテリアルは、ライトに反応する方法を示します。マテリアルの ***DiffuseColor*** はライトの ***DiffuseColor*** と反応します。拡散色はマットなベース色として理解できますが、鏡面反射色は照らされた表面から反射する色です。高い ***SpecularPower*** を使用すると、オブジェクトにメタリックな外観が与えられます。表面シリーズには ***ColorSaturation*** プロパティがあり、有効な範囲は 0～100%です。値を大きくすると、表面の塗りつぶしの色が強調され、シェーディング効果が減少します。

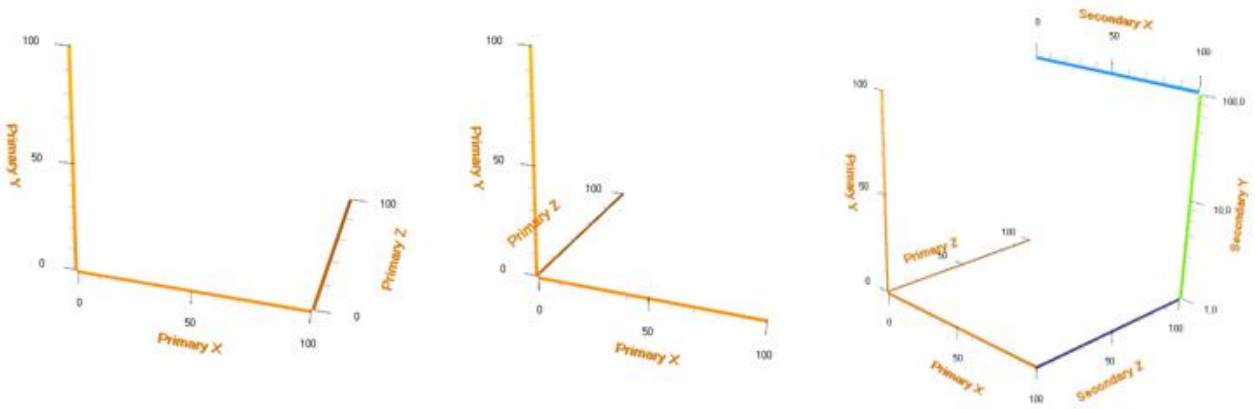




8.4 軸

各ディメンションにはプライマリとセカンダリの 2 つの軸があります。例えば、X ディメンションには、***XAxisPrimary3D*** および ***XAxisSecondary3D*** があります。一般的には、3D の軸は ViewXY's の軸と非常に良く似た動作を行います。

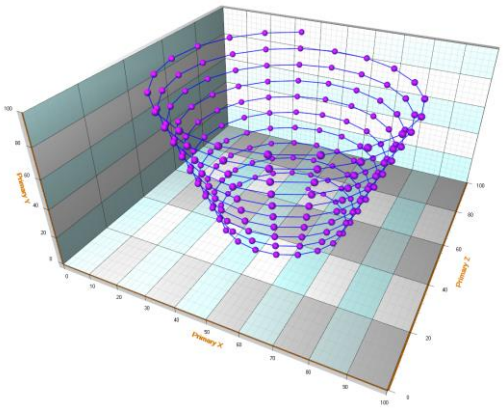
軸は 3D モデルボックスの角に配置できます。軸の ***Location*** プロパティを使用して位置を調整します。



8.5 一般の 3D シリーズ

View3D のシリーズではさまざまな方法と形式でデータを視覚化できます。すべての系列は軸の値の範囲にバインドされ、ディメンションごとに主軸または副軸にバインドするシリーズを選択できます。***XAxisBinding***、***YAxisBinding***、および***ZAxisBinding***プロパティを使用して設定します。

8.6 PointLineSeries3D



PointLineSeries3D では 3D 空間でポイントとラインを表示できます。ポイントには多くの基本的な 3D シェイプが用意されています。***LineVisible*** プロパティが ***true*** に設定されている場合、ポイントは線で接続されます。ポイントは、実際の 3D ポイント、または、2D 形状として表示できます。ラインは、シェーディングされた 3D ライン、または、1 ピクセル幅のヘアラインとしてレンダリングできます。シリーズに大量のデータがある場合、パフォーマンスの問題を回避するために ***LineOptimization = Hairline*** を設定することを推奨します。

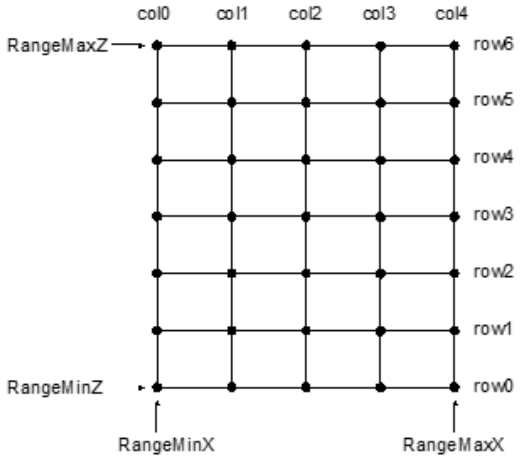
PointLineSeries3D は、次の 3 つの異なるポイント形式をサポートしています。Points プロパティ（SeriesPoint3D 配列）、PointsCompact プロパティ（SeriesPointCompact3D 配列）、PointsCompactColored プロパティ（SeriesPointCompactColored3D 配列）。***PointsCompact*** および ***PointsCompactColored*** 構造は非常にメモリ効率が高く、シンプルなポイントスタイルで最大 1 億個のデータポイントを視覚化できます。***PointsType*** プロパティを使用してポイント形式を設定します。

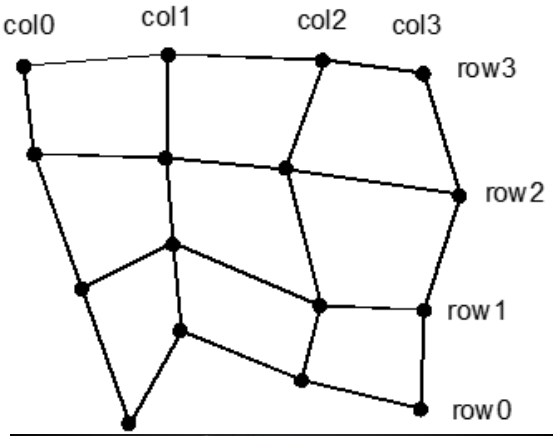
シリーズポイントをコードに追加する必要があります。 ***AddPoints(…)***メソッドを使用して、既存のポイントの最後にポイントを追加します。古いポイントを上書きしながら系列データ全体を一度に設定するには、新しいポイント配列を直接割り当てます。

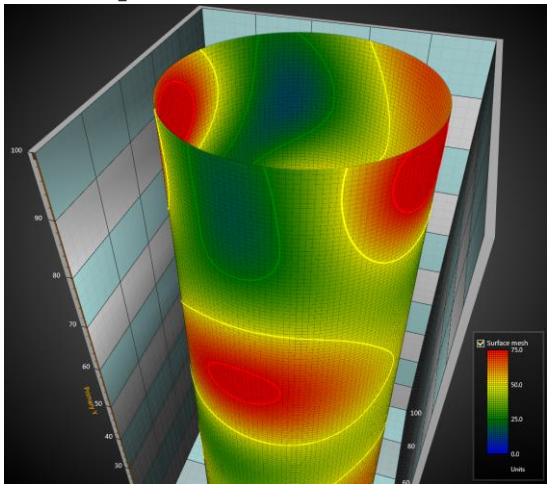
IndividualPointColors = True を設定すると、***Material.DiffuseColor*** の代わりにポイントのカラーフィールドが適用されます。特定のデータポイントの色で線に色を付けるには、***MultiColorLine = True*** に設定します。チャートは、隣接するポイント間の色のグラデーションを補間します。 ***IndividualPointSizes = True*** を設定すると、ポイントの ***sizeFactor*** フィールドが有効になります。この係数は、***PointStyle.Size*** で定義されたサイズを乗算します。

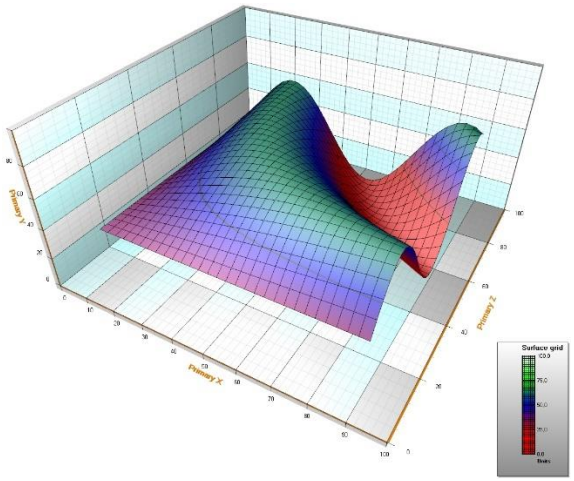
8.7 SurfaceGrid- および SurfaceMeshSeries3D

SurfaceGridSeries3D と ***SurfaceMechSeries3D*** では、データを 3D サーフェスとして視覚化できます。 ***SurfaceGridSeries3D*** では、ノードは X 次元と Z 次元で等間隔に配置されますが、***SurfaceMeshSeries 3D*** では、サーフェスノードは 3D スペースで自由に配置する事ができ、バーチャル上、あらゆる形に対しサーフェスをワーピングする事ができます。両シリーズともに、レンダリング等高線とワイヤーフレームのレンダリングを可能にします。









左図 IntensityGridSeries , 右図 IntensityMeshSeries

Fill プロパティを使用してサーフェスの塗りつぶしスタイルを選択します。***ContourPalette*** プロパティでは、高さの色付けの色ステップを定義できます。***ContourPalette*** はまた、***Wireframe mesh*** および ***Contour lines*** にも使用できます。
輪郭パレットのステップ数は無制限に定義できます。各ステップには、高さの値と対応する色があります。パレットは、***MinValue***、 ***Type***、そして ***Steps*** プロパティを含みます。***Type*** には、2 つの選択肢があります。すなわち、***Uniform*** および ***Gradient*** です。
表面グリッドデータの設定

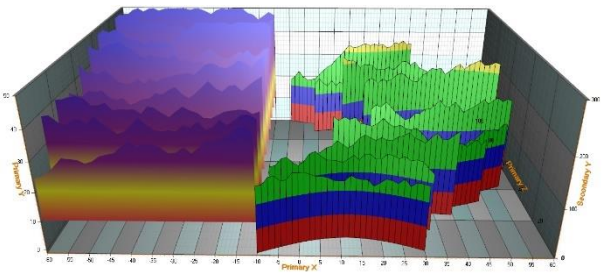
- ***RangeMinX*** および ***RangeMaxX*** プロパティを使用して X 範囲を設定し、割り当てられた X 軸に基づいて最小値と最大値を並べます。
- ***RangeMinZ*** および ***RangeMaxZ*** プロパティを使用して Z 範囲を設定し、割り当てられた Z 軸に基づいて最小値と最大値を並べます。
- ***SizeX*** および ***SizeZ*** プロパティを設定して、グリッドに列と行のサイズを与えます。
- すべてのノードの Y 値を設定します。

表面メッシュデータの設定

- ***SizeX*** および ***SizeZ*** プロパティを設定して、グリッドに列と行のサイズを与えます。
- すべてのノードの X, Y および Z 値を設定します。

グリッド/メッシュをリフレッシュするには、***InvalidateData()*** メソッドを呼び出します。

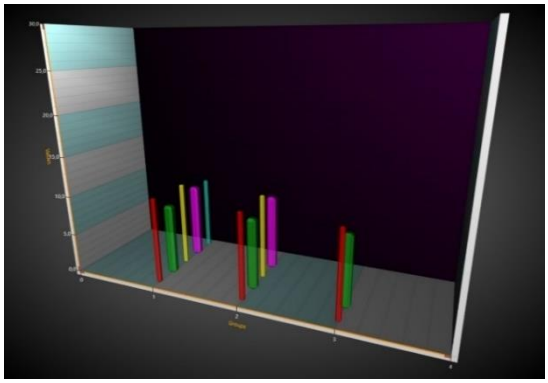
8.8 WaterfallSeries3D



WaterfallSeries3D では、データはエリアストリップで視覚化されます。領域は ***SurfaceGridSeries3D*** のように塗りつぶす、ワイヤーフレームで囲む、輪郭を描くことができます。Y 次元では、領域は ***BaseLevel*** プロパ

ティ値から始まります。ノードデータは、***SurfaceMeshSeries3D*** のように設定できます。***WaterfallSeries3D*** は、従来の 3D スペクトラムを表示するのに特に便利です。

8.9 BarSeries3D



BarSeries3D を使用すると、3D でバーデータの視覚化ができます。バーシリーズデータは、***x***、***y***、***z***、および**テキスト**フィールドを含む ***BarSeriesValue3D*** 構造として追加できます。***BarSeries3D*** には、バーの形状を設定するための ***Shape*** プロパティがあります。さらに、一部の形状では ***CornerPercentage*** を使用して角の丸みを変更し、***DetailLevel*** を使用して視覚的品質を変更できます。

バーシリーズは、View3D の ***BarViewOptions*** プロパティで使用可能な多くのオプションでグループ化できます。 ***BarViewOptions.ViewGrouping*** は、3D ビューでバーをグループ化する方法を設定します。

8.10 MeshModels

MeshModels リストプロパティを使用すると、外部 3D モデルエディターから LightningChart View3D に 3D モデルを挿入できます。モデルは OBJ 形式でインポートできます。OBJ 形式は、3D モデリングアプリケーションおよびゲームエンジンの一般的な形式です。***MeshModel*** の作成では .obj ファイルの頂点の色がサポートされます。頂点の位置は、x、y、および z（XYZRGBA) の後の赤、緑、青、およびアルファ値をサポートします。

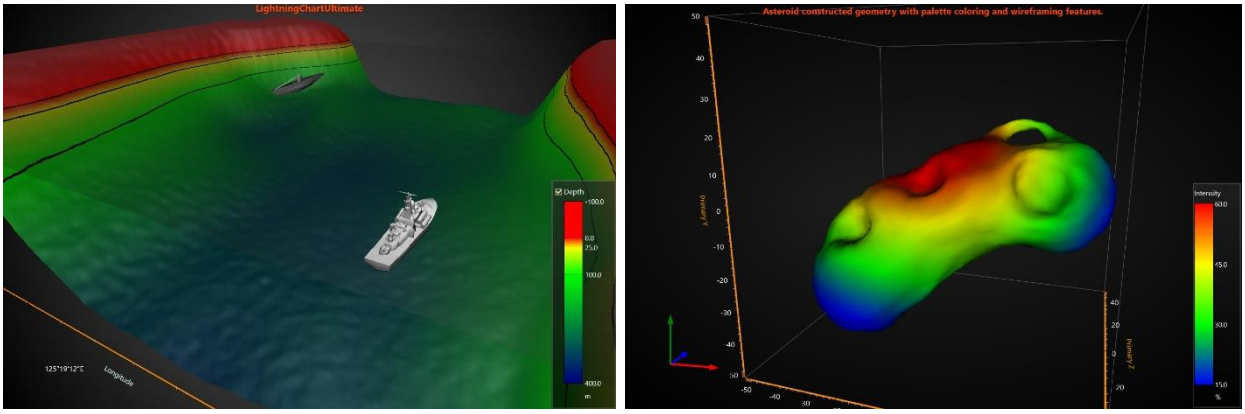
- ファイルからモデルをロードするには、***ModelFileName*** プロパティにパスとファイル名を設定するか、***LoadFromFile*** メソッドを使用します。ファイルからモデルをロード

するとき、テクスチャフィルも同じパスに存在する場合はロードされ、MTL ファイルと画像ファイルにアクセスできます。

- ストリームからモデルをロードするには、***LoadFromStream*** メソッドを使用します。ストリーム読み取りメソッドは、ジオメトリとマテリアルのみを読み取り、テクスチャは読み取りません。

- リソースからモデルをロードするには、***LoadFromResource*** メソッドを使用します。

MeshModel オブジェクトの ***Position*** は、割り当てられた X、Y、Z 軸に従います。 ***Rotation*** プロパティを編集してモデルを回転できます。***Size*** は ***Size*** プロパティで定義できます。これは元のモデルサイズの要因のコレクションであり、軸範囲または 3D ワールド寸法には従いません。



デフォルトでは、モデルは OBJ モデルの色でレンダリングされます。Fill を有効にして Fill を表示してください。ワイヤフレームを表示するには、***WireFrame*** = True に設定し、
WireFrameLineColor で優先線の色を設定します。
モデルの頂点にカスタムカラーを適用するには、モデルが作成された後で、***UpdateFillColors(int[]colors)***メソッドを使用します。***GeometryConstructed*** イベントを使用して、これをチェックします。***UpdateFillColors*** は、定期的に呼び出して、リアルタイムカラー更新を適用する事もできます。このメソッドは、頂点位置の長さが等しい ***(X Length)*** ARGB カラー配列が各頂点に 1 色必要です。
一部のモデルは逆巻き順で作成されているため、カリングによってモデルが見えなくなります。モデルが正しく表示されない場合は、***Cull*** 設定を ***Clockwise***、***CounterClockwise***、***None*** の間で変更します。

<u>頂点からプログラムで MeshModel を構築する</u>

MeshModel ジオメトリは、プロブラムでも構築する事ができます。回転、スケーリング、位置決めのプロパティなど、およびイベントは、ロードされたオブジェクトに対して機能するのと同様の方法で、頂点からプログラム作成された ***MeshModel*** にも適用されます。

次の ***Create*** メソッドを使用できます。

- *Create(positions, colors, indices)*

- *Create(positions, colors, normals, indices)*
- *Create(positions, textureCoordinates, bitmap, textureWrapMode, indices)*
- *Create(positions, normals, textureCoordinates, bitmap, textureWrapMode, indices)*

インデックス配列（***indices***）パラメーターはオプションです。指定すると、指定された配列から使用する頂点、色、ライト法線、テクスチャ座標を定義します。インデックスを使用すると、同じ頂点が複数の三角形で共有される場合にリソースが節約されます。

8.11 VolumeModels

VolumeModels は、Direct Volume Rendering によるボリュームデータの視覚化のためのツールです。***VolumeModel*** は内部のボリュームデータを取得して視覚化します。LightningChart のボリュームレンダリングエンジンは、***Volume Ray Casting*** に基づいています。

画像は、データセット内を移動する光線のトラックに沿ってサンプリングするボリュームデータを介してアルゴリズムによって生成されます。**Volume Ray Casting** のハードウェアアクセラレーションを簡単に実現するには、ボリュームオブジェクトの境界を生成する必要があります。通常それらはキューブで表されます。アーティファクトのない高いレンダリング品質、および交換可能なレイ機能の使用は、この技術の主な利点です。

RayFunction **はアルゴリズムのコアであり、非常に高いレベルの柔軟性を提供します。この手法はデータのサンプリング方法と結合方法を指定するため、強力です。これにより、特徴抽出に非常に便利なツールになります。**

注意：VolumeModels は、DirectX 11 レンダラーが使用されている場合にのみ使用できます。

データのロード

ロード関数とコンストラクターは、データをスライスのコレクションとして（Data プロパティと同様）、またはスライスを含むフォルダーへのパスを持つ文字列として提供することができます。データは、ツールで作成されたテクスチャマップとして提供することもできます。テクスチャマップはスライスで構成されていますが、その補足には写真上のスライスの数に関する追加情報も必要です。テクスチャマップは ***ChartTools.CreateMap*** 関数を介して作成できます。テクスチャマップの直接入力は、非常に大きなデータセットのアプリケーションの起動を高速化するために使用されます。データは、***LoadFromSlices()*** のようなロード関数の 1 つを介して、***VolumeModel*** へ供給する事ができます。

プロパティ

VolumeModel には、***Visible***、***Rotation***、***Size***、***Position***、***AllowUserInteraction***、***HighLight*** など LightningChart の 3D オブジェクトの一般的なプロパティが含まれています。さらに、オブジェクトには特定のプロパティがあり、Volume Rendering エンジンによるオブジェクトの処理方法を定義します。

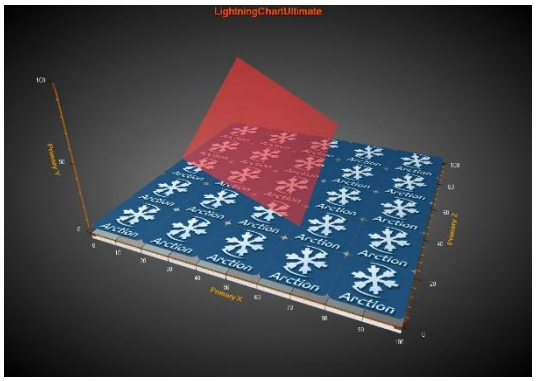
RayFunction プロパティを使用すると、LightningChart Volume Rendering Engine で使用可能なボクセルサンプリングと合成の 3 つの方法のいずれかを選択できます。

- ***RayFunction.Accumulation***は可能な限り多くのデータを収集および結合します。この手法によって生成される視覚化は、半透明のゲルのように見えます。以下の図は、医療データセットを視覚化する***RayFunction.Accumulation***アプリケーションの例を示しています。
- ***RayFunction.MaximalIntensity***は光線によってサンプリングされた最も明るい値のみを考慮します。視覚的にはX線画像と非常によく似た結果が得られ、またオブジェクトの内部構造に関する追加情報を取得できます。スケルトンの可視化と超音波の干渉シミュレーションのための***RayFunction.MaximalIntensity***アプリケーションを以下に示します。
- ***RayFunction.Isosurface*** は、ポリゴンサーフェスのレンダリングのようにモデルサーフェスを描画します。**Indirect Volume Rendering**によって生成される結果と非常に似ています。図は、人間の頭蓋骨CTの視覚化と水流のシミュレーションのための ***RayFunction.Isosurface***アプリケーションの例を示しています。

ボリュームレンダリングエンジンは、プロパティによって**しきい**値範囲を ***VolumeModel*** に適用できます。すべてのカラーチャンネルに個別の境界があります。ボクセルは、対応するカラー値がすべてのチャンネルで高い境界より低く、低い境界より高い場合にのみ視覚化されます。許容範囲は見えません。このプロパティは、マウスヒットテストでは考慮されません。

SliceRange プロパティにより、***VolumeModel*** の一部を切り取ることができます。これは、オブジェクトの内部構造を調べるための非常に便利なツールです。 ***SliceRange*** には ***Min*** と ***Max*** の 2 つの境界が含まれ、どちらも 3 つのポインティングフロート値で表されます。

8.12 Rectangle3D オブジェクト

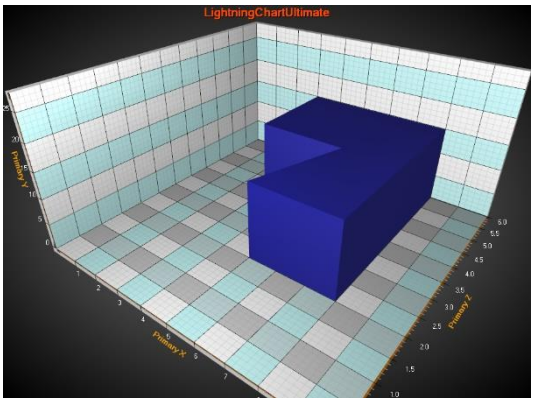


Rectangle3D を使用すると、任意の角度、任意のサイズ、任意の場所で回転した長方形を表示できます。***View3D.Rectangles*** リストに追加できます。長方形は、***View3D.Dimensions*** に従ってサイズを定義することにより平面として機能することもできます。

3D ワールド寸法（X、Y、Z 軸値ではない）の ***Size*** を ***Width*** と ***Height*** として設定します。X、、Y、Z 軸の値で定義された ***Center*** プロパティで中心点を設定します。 ***Rotation*** プロパティは、回転を度単位で指定します。

塗りつぶし設定は、***Fill*** プロパティで変更できます。単色およびビットマップの塗りつぶしが利用可能です。ビットマップの塗りつぶしを使用するには、***Image*** でビットマップを設定し、***UseImage*** を有効にします。 ***Fill.Layout = Stretch*** を設定すると、ビットマップが拡大されて四角形が塗りつぶされます。***Fill.Layout = Tile*** を設定することにより、同じビットマップがタイル状に並べられて四角形が塗りつぶされます。タイルカウントは、***Fill.TileCountWidth*** および ***Fill.TileCountHeight*** プロパティを介して変更できます

8.13 Polygon3D オブジェクト



Polygon3D オブジェクトを使用すると、指定された Y 範囲に引き伸ばされた 2D ポリゴンを表示できます。 ***View3D.Polygons*** リストに追加できます。

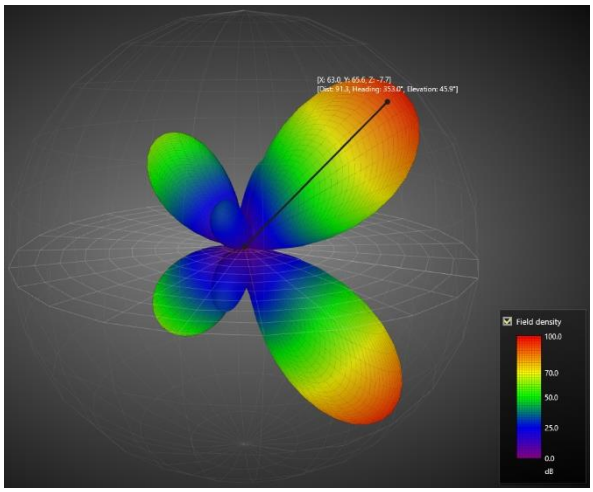
X および Z 軸の値でポリゴンパスを定義し、***Points*** 配列に保存します。 Y 範囲を ***YMin*** および ***YMax*** 値で設定します。***Material.Diffuse*** は、四角形のメインカラーを設定します。度単位で ***Rotation.X***、***Rotation.Y***、および ***Rotation.Z*** を使用して、ポリゴンを別の角度に回転させます。

8.14 座標軸コンパーター

次の座標系コンバーターは View3D の使用を補完する ***CoordinateConverters*** 名前空間で使用できます。

- デカルト 3D <->球面 3D
- デカルト 3D <->円筒 3D

SphericalCartesian3D



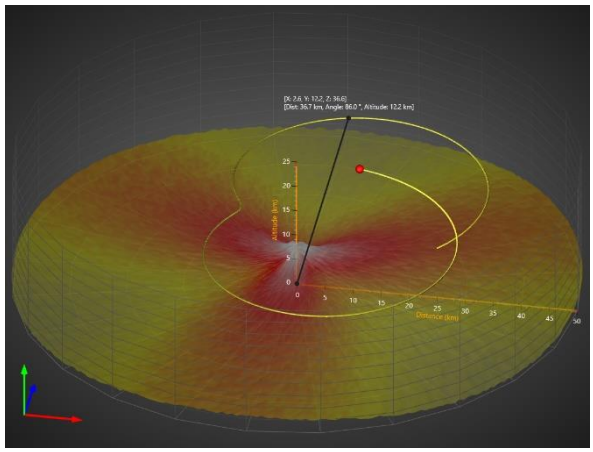
SphericalCartesian3D コンバータークラスは、球面座標と 3D デカルト座標を変換します。

球状のデータポイントは、次のフィールドを含む ***SphericalPoint*** オブジェクトによって定義されます。

- ***Distance*** 原点からの距離（0,0,0）
- ***ElevationAngle*** 仰角。標高または高度とも呼ばれ、XZ 平面から測定されます。ElevationAngle は 90 度 - 傾斜角です。XZ 平面は、レフェレンス平面です。
- ***HeadingAngle*** 方位角。方位角および絶対方位角とも呼ばれます。

CylindricalPoint をデカルト座標に変換するには、***SphericalCartesian3D.ToCartesian()*** メソッドを使用します。デカルトポイントを球面ポイントに変換するには、***SphericalCartesian3D.ToSpherical()*** メソッドを使用します。

CylindricalCartesian3D



円筒座標と 3D デカルト座標を変換するコンバータークラスです。

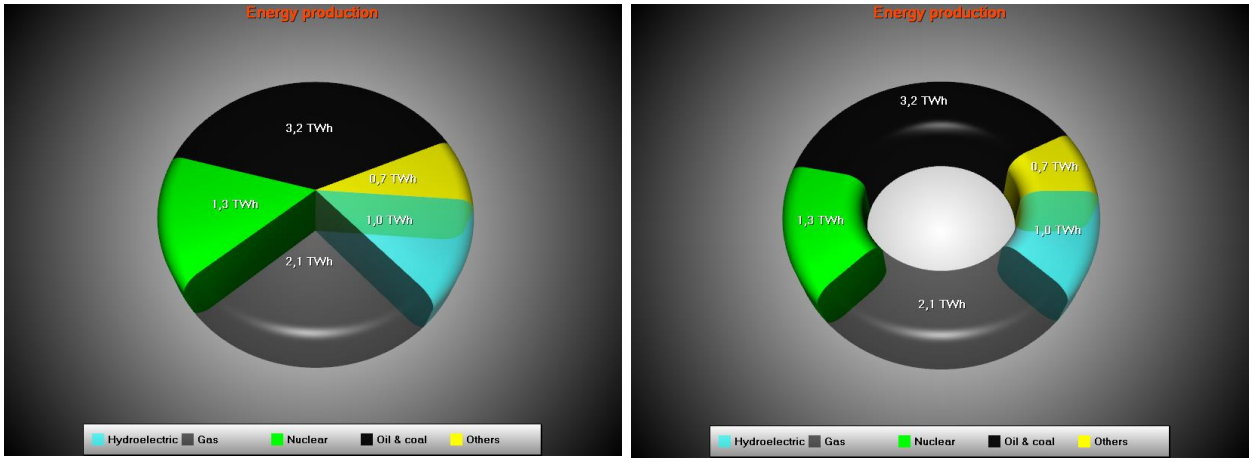
円筒ポイントは、次のフィールドを含む ***CylindicalPoint*** オブジェクトによって定義されます。

- ***Distance*** XZ 平面に沿った距離
- ***Y***: Y 値
- ***Angle*** 方位角および絶対方位とも呼ばれる方位角

CylindricalPoint をデカルト座標に変換するには、***CylindricalCartesian3D.ToCartesian()*** メソッドを使用します。デカルトポイントを円柱ポイントに変換するには、***CylindricalCartesian3D.ToCylindrical()*** メソッドを使用します。

9. ViewPie3D

ViewPie3D は、データを円グラフとドーナツグラフとして 3D で表示します。



Style プロパティを使用して、グラフの種類（***Pie*** または ***Donut***）を選択します。View3D と同様に、***ZoomPanOptions*** プロパティツリーでズーム、パン、および回転を制御します **(Error! Reference source not found** 章を参照）。

Camera プロパティは視点を制御します（第 8.2 章を参照）。***LightingScheme*** プロパティで定義済みの照明設定を選択できます。***Material*** プロパティとそのサブプロパティを使用して、一般的な 3D サーフェスの外観と光沢を調整します。

DonutInnerPercents を使用してドーナツの内側半径を設定し、***Rounding*** を使用してエッジの丸みの半径を調整し、***StartAngle*** を使用してパイを回転させ、***Thickness*** を使用してパイの厚さを調整します。***ExplodePercents*** は、スライスの ***Explode*** が ***true*** に設定されている場合、爆発したパイスライスの距離を調整します。

TitlesStyle は、***Titles***、***Values***、または ***Percents*** のいずれかのパイスライステキストを設定します。***TitlesNumberFormat*** を編集して、たとえば「0.0 TWh」にして、最後に単位を含めます。

パイスライス

円グラフのデータは ***Values*** コレクションに保存されます。リスト内の各アイテムのタイプは ***PieSlice*** です。***Value*** プロパティのデータ値を編集し、タイトル文字列を ***Title.Text*** プロパティに設定します。***TitleAlignment = Outside*** を定義するとタイトルは円の外側に描画されます。

コードでのパイスライスの設定

```
PieSlice slice1 = new PieSlice("Hydroelectric",
Color.FromArgb(150, Color.Aqua), 1.0, chart.ViewPie3D, true);
```

10. ViewPolar

ViewPolar を使用すると、極形式でデータを視覚化できます。データポイントの位置は、角度値と振幅によって決定されます（ViewXY で角度を X、振幅を Y と比較してください）。ポーラービューには、ズーム機能とパン機能もあります。

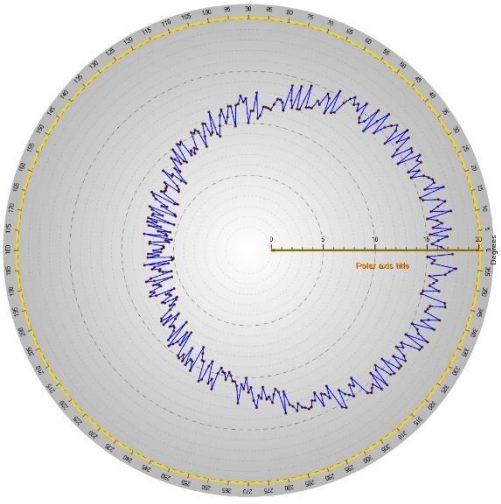
10.1 軸

極座標軸は、***Axes*** リストプロパティを介して定義できます。同じチャートで複数の軸を使用できます。シリーズの ***AssignPolarAxisIndex*** プロパティを設定することにより、これらの軸のいずれかにシリーズを割り当てることができます。軸は角度スケールと振幅スケールの両方を表します。それ以外は、極軸は ViewXY 軸に非常に似ています。

AngleOrigin を使用して、角度スケールの回転角度を設定します。***AmplitudeAxisAngle*** を使用して、振幅軸の位置を回転させます。振幅スケール角度は、絶対角度（***AmpitudeAxisAngleType = Absolute***）または角度スケールの角度に対する相対角度（***AmpitudeAxisAngleType = Relative***）として設定できます。

MajorDivCount で振幅除算カウントを設定し、***MajorDiv*** プロパティで除算の大きさを設定します。振幅スケールはそれに応じて調整（***MaxAmplitude*** が更新）されます。 ***MinorDivCount*** で振幅のマイナー除算カウントを設定します。デフォルトでは、チャートは可能な限り多くの角度区分を含めようとします。角度分割を制御するには、***AngularAxisAutoDivSpacing*** を ***False*** に設定します。次に、チャートは部門の ***AngularAxisMajorDivCount*** カウントを試行します。グラフのスペースが小さすぎてすべての区分とラベルをレンダリングできない場合、それが収まるほど低い区分カウントを使用します。

10.2 PointLineSeriesPolar



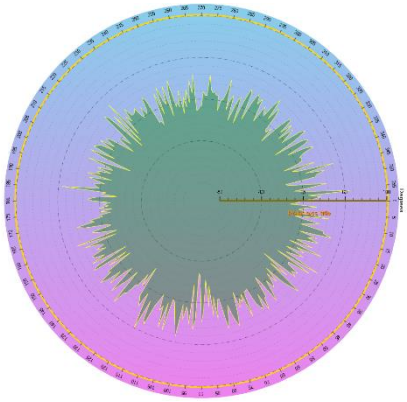
ViewPolar の ***PointLineSeriesPolar*** を使用して、線、点のグループ、または点線を描画できます。多くの線と点のスタイルは、***LineStyle*** プロパティと ***PointStyle*** プロパティで使用できます。

ラインカラーリングはパレットをサポートしています。***ColorStyle*** プロパティを使用して、パレットの色の適用方法を選択できます。

- ***LineStyle*** パレットの塗りつぶしなし。***LineStyle.Color*** プロパティで設定された色が適用されます。
- ***PalettedByAngle*** データポイントの ***Angle*** フィールドが色を決定します。
- ***PalettedByAmplitude*** データポイントの ***Amptitude*** フィールドが色を決定します。
- ***PalettedByValue***: データポイント ***Value*** フィールドが色を決定します。

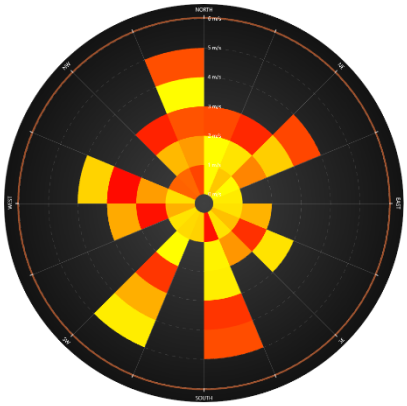
ValueRangePalette プロパティを使用して、色と値のステップを定義します。ViewXYs および View3D のシリーズと同様に機能します。

10.3 AreaSeries



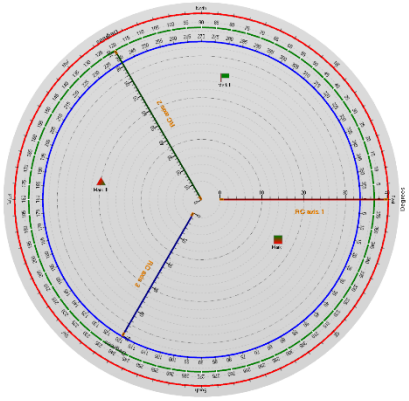
エリアシリーズでは、塗りつぶされたエリアスタイルでデータを視覚化できます。ラインに色が塗られた内側/下側のエリアを除き、PointLineSeries と同じように機能します。エッジの線スタイルは、***LineStyle*** プロパティで編集できます。***FillColor*** プロパティで塗りつぶしを変更できます。

10.4 セクター



Sectors は、角度または振幅の範囲を示すように定義できます。***MinAmplitude*** プロパティと ***MaxAmplitude*** プロパティで振幅範囲を定義します。***BeginAngle*** と ***EndAngle*** で角度範囲を定義します。マウスでセクターをドラッグして移動させます

10.5 マーカー



マーカーを使用して特定の位置で特定のデータ値をマークできます。***AssignPolarAxisIndex*** を設定して、マーカーに優先軸を割り当てます。***Amplitude*** および ***AngleValue*** プロパティを定義して、配置します。***Symbol*** を編集して好みの外観にし、***Label*** プロパティでマーカーテキストを定義します。

マーカーはマウスでドラッグして移動できます。***SnapToClosestPoint*** を ***Selected*** または ***All*** に設定して、ドラッグ時に最も近いデータポイントのスナップを有効にします。***Selected*** はこのマーカーが ***SetSnapSeries()***メソッドでスナップするように設定されているシリーズのみが追跡されます。***All*** はすべてのシリーズを追跡します。

11. ViewSmith

スミスチャートは、一般にインピーダンス測定およびインピーダンス整合アプリケーションの電子機器で使用されます。スミスチャートは、データを実数値と虚数値（**R + jX**）でプロットします。

定義

インピーダンス = Z = R + jX

R = レジスタンス、実数部分

X = リアクタンス、虚数部分

X > 0: 容量性
X < 0: 誘導性

データの位置は円形の実数および虚数の対数目盛の角度によって2Dプロットで決定されます。

11.1 軸

スミスチャートには実際の軸が1つしかありません。これは、拡張プロパティツリーの***Axis***を介して構成できます。プロパティのほとんどは、ViewPolarの軸およびViewXYの軸と同一です。また、ViewSmith 調整に固有の高度なプロパティもあります。

GridDivCount は、実軸の円形グリッド線と虚数スケールの対数グリッド線の量を定義します。

GridImg および***GridReal***プロパティは、**実数**スケールまたは**虚数**スケールでグリッド線をカス

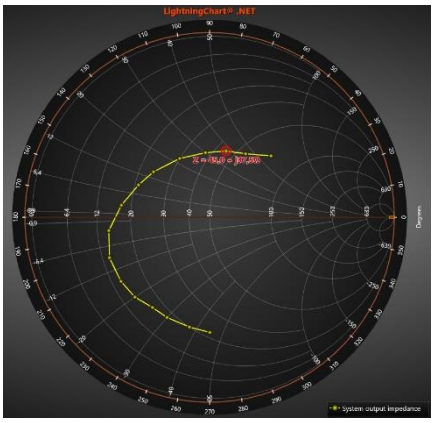
タマイズします。さらに、***Visible***プロパティを使用してグリッドを非表示にできるため、ユーザーはグリッドの1つを非表示にして、引き続き別のグリッドで作業できます。

RealAxisLineVisible プロパティは、軸線を非表示にします。

ShowAbsoluteValues プロパティはスケール上の値（絶対値または正規化）を定義します。

ClipGridInsideGraph は、グラフ線の外側にグリッド線を表示します。

11.2 PointLineSeries



ViewSmith の ***PointLineSeries*** を使用して、ViewPolar のように線、点のグループ、または点線を描画できます。多くの線と点のスタイルは、***LineStyle*** と ***PointStyle*** プロパティで使用できます。

以下のコードでスミスチャートのコレクションに1セットのデータポイントを追加します。

```
SmithSeriesPoint[] m_aPoints;
PointLineSeriesSmith Series = new PointLineSeriesSmith(m_chart.ViewSmith, axis);
//Create data for series
m_iCount = 5000;
m_aPoints = new SmithSeriesPoint[m_iCount];
for (int i = 0; i < m_iCount; i++)
{
          // Sine from left to right
          m_aPoints[i].RealValue = i * (MaxReal / m_iCount);
          m_aPoints[i].ImgValue = Math.Sin(0.01 * i)/Math.PI * MaxReal;
}
Series.Points = m_aPoints;
//Add series to chart
m_chart.ViewSmith.PointLineSeries.Add(Series);
```

11.3 マーカー

マーカーを使用して特定の位置で特定のデータ値をマークできます。マーカーはマウスでドラッグして移動できます。このプロパティにはViewPolarのマーカーと同じ定義があります (第10.8章参照)。

ImgValueおよび***RealValue***プロパティを定義して配置します。***Symbol***を編集して好みの外観にし、***Label***プロパティでテキストを定義します。

12. 凡例ボックス

ViewXY は、複数の凡例ボックスを同じグラフでサポートします。これらの凡例ボックスを ***ViewXY.LegendBoxes*** コレクションに挿入します。他のビューでの凡例ボックス (3D polar, smith, pie) は、ViewXY シリーズの凡例ボックスと酷似しています。しかし、1 つのグラフにつき、1 つの凡例ボックスしか認められていません。また、セグメントベースのプロパティは、axes がセグメントへ分割できないため存在しません。

ShowCheckBoxes プロパティを使って、凡例ボックスでチェックボックスを表示したり非表示にしたりできます。チェックボックスの外観を変えるには、***CheckBoxColor*** と ***CheckMarkColor*** を使い、***CheckBoxSize*** プロパティは、ピクセルでボックスのサイズを調整します。

```
_chart.ViewXY.LegendBoxes[0].ShowCheckboxes = true;
_chart.ViewXY.LegendBoxes[0].CheckBoxColor = Colors.Green;
_chart.ViewXY.LegendBoxes[0].CheckMarkColor = Colors.Blue;
_chart.ViewXY.LegendBoxes[0].CheckBoxSize = 15;
```

ShowIcons = False を設定してアイコンを非表示にできます。特定のシリーズを凡例ボックスのリストに掲載しない場合は、 ***series.ShowInLegendBox = False*** と設定してください。***IntensityScales.Visible = False*** を設定して、***IntensityGrid*** または ***-Mesh*** のパレットスケールが非表示にできます。***ScaleSizeDim1*** および ***ScaleSizeDim2*** プロパティを設定してサイズ変更ができます。タイトルの位置とスケールのボーダーも、変更する事ができます。View3D は、ViewXY's ***IntensityScales*** の替わりに ***SurfaceScales*** プロパティを持っている事を覚えておいてください。

位置の制御

凡例ボックスは自動または手動で配置できます。自動配置により、グラフのセグメントの左側、上部、右側、下部、またはマージンに配置できます。***Position*** プロパティで位置を制御します。配置オプションは、***TopCenter***, ***TopLeft***, ***TopRight***, ***LeftCenter***, ***RightCenter***, ***BottomLeft***, ***BottomCenter***, ***BottomRight***, ***Manual*** となります。

ビューが数個のセグメントに分割される場合は、凡例ボックスが属するセグメントに対してそれを調整する事ができます。(SegmentIndex を使います。)　セグメントを基本とする制御には ***SegmentTopLeft***, ***SegmentTopCenter***, ***SegmentTopRight***, ***SegmentBottomLeft***, ***SegmentBottomCenter***, ***SegmentBottomRight***, ***SegmentLeftMarginCenter***, ***SegmentRightMarginCenter*** のオプションがあります。

Offset プロパティは、***Position*** プロパティで決定された位置から指定された量だけ位置をシフトします。

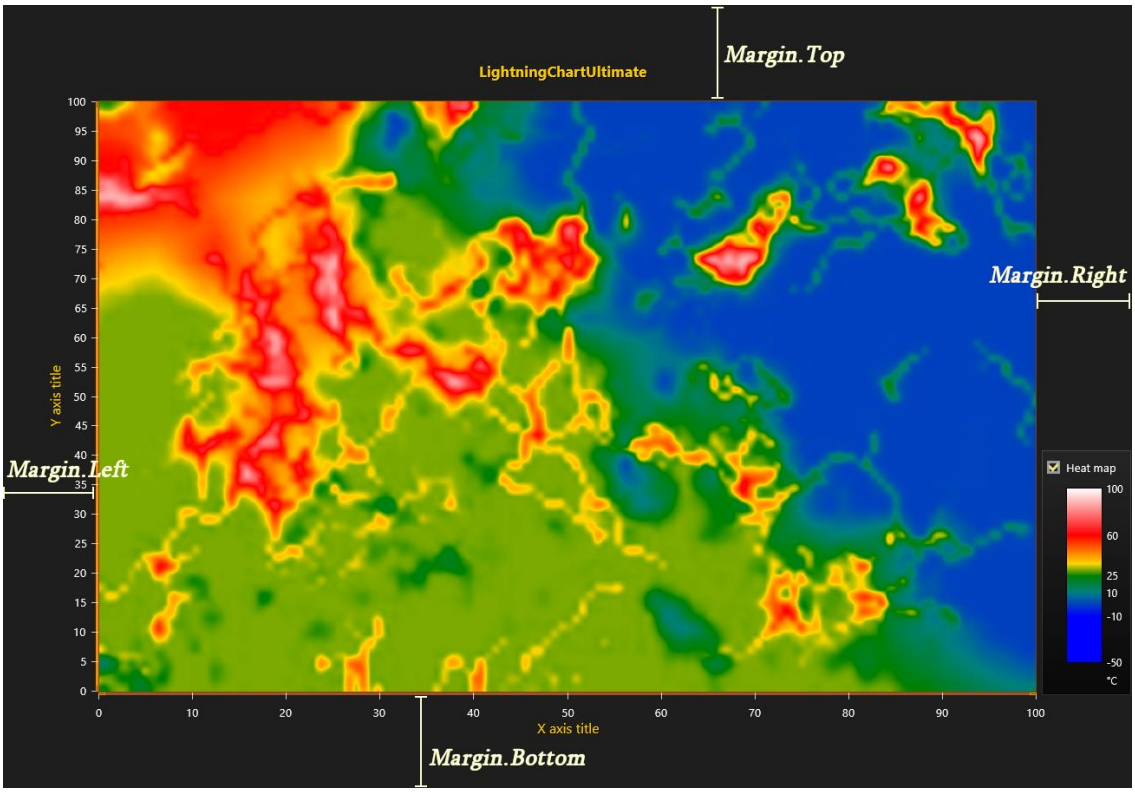
```
// Setting legend box position, offset shifts from RightCenter position
 chart.ViewXY.LegendBoxes[0].Position = LegendBoxPositionXY.RightCenter;
 chart.ViewXY.LegendBoxes[0].Offset = new PointIntXY(-15, -70);
```

Manual 配置では、凡例ボックスの左上隅からビューの左上隅までのオフセットが計算されます。これは、グラフ領域の上部から計算される ***TopLeft*** オプションとは異なります。凡例ボックスを移動する場合、またはサイズを変更する場合、その ***Position*** は ***Manual*** に設定され、***Offset*** プロパティは、新しい配置を繁栄するために更新される事を覚えておいてください。

自動凡例ボックスのアライメントは、「***Manual***」以外のオプションに ***Position*** を設定して戻すまでは無効になります。***Position*** オプション間で切り替える場合は、***Offset*** は更新されませんので、凡例ボックスが消えるように見える事もあります。（ビューの外側に位置します。）***Offset*** を 0.0 に設定して戻しこれを直します。

13. マージン

マージンはグラフエリアの空のスペースです。ビューのコンテンツは、マージンの外側で自動的にクリップされます。画面の座標で位置が定義されているため、グラフのタイトル、注釈、凡例ボックス以外のすべてのコンテンツがクリップされ、余白にも自由に配置できます。1 ピクセル幅のボーダー長方形 ***Border*** を描画して、マージンがある場所を表示できます。四角形の色は、***Border.Color*** を使用して変更できます。デフォルトではボーダーは ***ViewXY*** だけに表示されます。



AutoAdjustMargins が ***Enabled*** になっている場合、グラフのサイズはすべての軸とグラフのタ イトルに十分なスペースがあるように調整されます。***Disabled*** にすると ***Margins*** プロパティが適 用され、マージンを手動で設定できます。

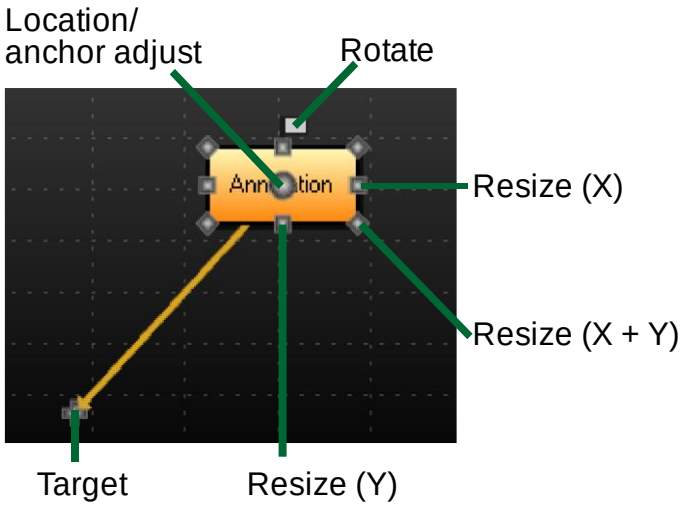
実行時に、***GetMarginsRect*** メソッドを呼び出すことにより、マージンの長方形をピク セル単位で取得できます。これは、自動マージンと手動マージンの両方に適用されます。画面 座標ベースの計算またはオブジェクト配置を行う必要がある場合に役立ちます。

MarginsChanged イベントは、たとえばサイズ変更のためにマージンの長方形が変更されたときにトリガーするように設定できます。

14. 注釈

Annotations は、マウスインターラクティブテキストラベル、または、グラフィックスをチャートのどの場所でも表示する事を可能にします。Annotations は、マウスで移動、サイズ変更、回転させる事ができ、そのターゲットと場所などが変更できます。あるいは、Annotations は、コードで管理する事もできます。Annotations は、様々なスタイルや形状でレンダーする事が可能ですので、画面上でカスタムグラフィックスがレンダーされなければならない場合も良い働ききをします。***AnnotationXY*** オブジェクトを ***ViewXY.Annotations*** コレクションに作成します。

マウスを annotation の上で動かすと、マウスでインターアクティブ編集できる状態になり、annotation、サイズ変更、回転、矢印が指し示す方向を決定する事が可能になります。



14.1.1 ターゲットと場所の設定

Target は、矢印の終点、つまり矢印または吹き出しの先端が指す点です。***Target*** は、軸値または画面座標で設定できます。***TargetCoordinateSystem*** を使用して、***AxisValues*** または ***ScreenCoordinates*** を選択します。***AxisValues*** が選択されている場合、***TargetAxisValues*** プロパティは矢印線が指す場所（矢印線の端）を設定します。***TargetScreenCoords*** を使用すると、画面座標で設定できます。

Location は矢印の始点です。画面座標、軸の値、または ***Target*** からの相対オフセットとして設定できます。場所を決めるには ***LocationCoordinateSystem*** を使用して選択し、***LocationScreenCoords***、***LocationAxisValues*** または ***LocationRelativeOffset*** を使用して、設定します。***Location*** は、テキスト領域の回転の中心点でもあります。

Anchor プロパティは、テキスト領域を ***Location*** に配置する方法を設定します。***Anchor.X*** = 0.5、***Anchor.Y*** = 0.5 を設定すると、矢印の始まりが中央になります。***Anchor.X*** = 0. 1 および Anchor.Y = 0.25 を設定すると、次の図に示すように、矢印の開始点が左上隅近くになります。

14.1.2 マウスを使用した移動、回転、サイズ変更

Target からドラッグして、矢印の端を移動します。テキスト領域からドラッグして、新しい ***Location*** を設定します。丸い location/anchor ノードからドラッグすることにより、***Anchor*** と ***Location*** のプロパティを同時に調整し、テキストボックスを同じ場所に保持できます。

Shift キーを押しながら X または Y サイズ変更ノードからドラッグすると、対称操作が使用され、両側が同時に調整されます。***Shift*** キーを押しながらコーナーのサイズ変更ノード（X + Y）からドラッグすると、サイズ変更によりアスペクト比が維持されます。回転操作では、***Shift*** キーで回転角度を最も近い 15 度の倍数にスナップできます。

14. 1.3 外観の調整

Style プロパティを設定して、注釈の形状を選択します。選択肢は、***Rectangle***、***RectangleArrow***、***RoundedRectangle***、***RoundedRectangleArrow***、***Arrow***、***Callout***、***RoundedCallout***、***Ellipse***、***EllipseArrow***、***Triangle***、***TriangleArrow*** です。

矢印付きのスタイルでは、***ArrowLineStyle***、***ArrowStyleBegin***、および ***ArrowStyleEnd*** を使用して矢印のデザインを設定します。矢印の終端スタイルには、***None***、***Square***、***Arrow***、***Circle***、***Caliper*** があります。

Fill を使用して、注釈の塗りつぶしを変更します。編集状態のマウスインタラクティブノードの外観は、***NibStyle*** から変更できます。***TextStyle*** は、テキスト領域内のフォント設定とテキストの配置を設定します。***BorderLineStyle*** および ***CornerRoundRadius*** は、境界線の外観を設定します。

Sizing プロパティは、注釈テキストボックスのサイズを設定します。

- ***Automatic*** はコンテンツによってサイズを調整し、***AutoSizePadding*** スペースを境界線に残します。
- ***AxisValuesBoundaries*** で注釈のサイズを軸の値で設定できます。***AxisValuesBoundaries.XMin***、***XMax***、***YMin***、および ***YMax*** を使用して定義します。
- ***ScreenCoordinates*** は、画面座標による設定を可能にします。***SizeScreenCoords.Height*** と ***Width*** を使用します。

Annotation3D コレクションにより 3D シーンに注釈を追加できます。一般に、X、Y、Z ディメンションを使用する ***Target*** と ***Location*** のプロパティを除き、ViewXY の ***Annotation*** と同様です。***Target*** は、3D でマウスで移動できます。移動を補助するために、マウスが ***Target*** ノード上にあるときに注釈に十字線が表示されます。***ShowTargetCrosshair*** プロパティを ***Auto / On / Off*** に設定し、***TargetCrosshairLineStyle*** でラインスタイルを調整します。

ViewPolar と ViewSmith においては、***Annotation*** は、極座標軸 (角度と振幅)あるいはスミス軸の値 (実数と虚数)で定義される ***Target*** と ***Location*** を除いて View XY シリーズの***注釈***と似ています。 軸の値によるサイジングは適切ではないため、***Sizing*** プロパティの値は、Automatic と ScreenCoordinates のみです。

15. エクスポートと印刷

画像のエクスポート

SaveToFile()メソッドを使用して、グラフを.PNG、.BMP、および.JPG ファイルとしてエクスポートできます。 ***SaveToFile()*** メソッドを使用すると、解像度を下げて画像ファイルをエクスポートし、スムージング/アンチエイリアスオプションを使用できます。ストリームにエクスポートするには、***SaveToStream()***メソッドを使用します。

ViewXY、ViewPolar、および ViewSmith は、.WMF、.EMF、および.SVG 形式としてエクスポートすることもできます。 View3D および ViewPie3D は現在サポートしていません。選択したベクターファイル形式で ***SaveToFile*** または ***SaveToStream*** メソッドを使用します。

注意：ベクトル出力は単純化されており、複雑なポイントスタイルなどのすべての詳細は単純な色と単純な形状として表示できます。ベクトル出力にはビットマップ要素も含まれる場合があります。

クリップボードにコピー

CopyToClipboard(…)を呼び出して、チャートをクリップボードにコピーできます。 ViewXY、ViewPolar、および ViewSmith は、***CopyToClipboardAsEmf()***メソッドを使用してベクター形式でコピーできます。

バイト配列へのキャプチャ

チャートには ***CaptureToByteArray*** メソッドがあり、外部コンポーネントへの高速生画像データコピーまたはデータのさらなる処理として取得します。

連続フレーム書き込み用の出力ストリームの設定

chart.OutputStream プロパティを使用して、チャートがレンダリングフレームを書き込むストリームを設定します。このプロパティは特に ***Headless mode*** でチャートから連続フレームをキャプチャする最速の方法として意図されています（第 24 章を参照）。

ストリームは生のバイトストリームで、チャネルごとに 1 バイトで各ピクセル 4 バイトで記述されます。チャネルの順序は、レンダラーとその設定によります。生成される画像サイズは、ピクセル単位のチャートのサイズにする必要があります。***GetLastOutputStreamFormat*** メソッドと ***GetLastOutputStreamSize*** メソッドを使用して、最後に書き込まれた画像の形式と出力サイズを確認します。

注意： LightningChart の他のプロパティとは対照的に、チャートの破棄ではセットストリームは破棄されません。

印刷

PrintPreview() メソッドを呼び出して印刷プレビューダイアログを開くか、***Print()*** を呼び出してデフォルト設定で直接印刷します。手動設定で印刷するには ***Print(…)***を呼び出します。ViewXY、ViewPolar、ViewSmith の印刷は、ベクター印刷もサポートしています。***Print(…)***メソッドのパラメーターに ***Roaster*** または ***Vector*** 形式を設定します。

16. チャート更新

プロパティまたはシリーズのデータ値が変更されるたびに、LightningChart コントロールが再描画され、CPU とディスプレイアダプターのオーバーヘッドが発生します。プログラムで複数のプロパティを同時に変更する場合は***BeginUpdate()*** と ***EndUpdate()***メソッドの呼び出しの間にプロパティを変更する必要があります。***BeginUpdate()*** は ***EndUpdate()***が呼び出されるまでコントロールの描画を停止します。保留中の ***BeginUpdate()*** 呼び出しには内部カウンターがあり、同量の ***EndUpdate()*** 呼び出しに到達すると ***EndUpdate()*** はコントロールを再描画します。次の例はコンピューターへの負荷を最小限に抑えてチャートを更新する方法を示しています。

```
chart.BeginUpdate(); //Disable redraws

//Add data to series
chart.ViewXY.SampleDataSeries[0].AddSamples(multiChannelSampleStream[0],
    false);
chart.ViewXY.SampleDataSeries[1].AddSamples(multiChannelSampleStream[1],
    false);
chart.ViewXY.SampleDataSeries[2].AddSamples(multiChannelSampleStream[2],
    false);

//Update point counter bar
chart.ViewXY.BarSeries[0].SetValue(0,1,(double)totalPointsCollected,"",
    false);

// Point counter label
chart.Title.Text = totalPointsCollected.ToString();

// Set monitoring scroll position to latest x
newestX = firstSampleTimeStamp + (double)(pointsLen - 1) / genSampFreq;
chart.ViewXY.XAxes[0].ScrollPosition = newestX;

chart.EndUpdate(); // Enable redraws and redraw
```

シリーズデータの更新は、データの保存方法によって異なります。配列シリーズの場合、配列の内容を更新した後に ***InvalidateData()***メソッドを呼び出す必要があります。そうしない場合、変更について UI に通知されません。

ObservableCollection は、WPF の半バインド可能および完全バインド可能のデータバインドに役立ち、自動通知によりすべてのポイントまたはポイントのフィールドで自動的に更新されます。したがって、***InvalidateData()*** は必要ありません。ただし、ObservableCollections を使用すると配列またはリストと比較してパフォーマンスがわずかに低下します。これは、特に大量のデータポイントがある場合に顕著です。

17. LightningChart 通知、エラーおよび例外処理

バージョン 8.4 以降、LightningChart は ***ChartMessage*** イベントを通じてチャートからユーザーにメッセージを送信します。メッセージには、チャートのパフォーマンス、誤った使用法、警告、エラーなどに関する通知が含まれます。メッセージを確認するには ***chart.ChartMessage*** イベントハンドラーを定義します。イベントにはメッセージの情報を保持する ***ChartMessageInfo*** 構造が含まれます。

バージョン 8.4 より前では、***ChartError*** イベント（現在は廃止とマークされています）を介してメッセージを送信しますが、含まれる情報は ***ChartMessage*** よりも少なくなります。ユーザーは ***ChartMessages*** の代わりに ***ChartError*** イベントから同じ基本情報を取得できますが、***ChartMessage*** を使用することを推奨します。

ChartMessageInfo の ***MessageSeverity*** プロパティは、メッセージの重大度を示します。メッセージは、重大度に基づいてフィルタリングできます。メッセージの可能な重大度レベルは次のとおりです。

- ***Debug*** - 通常はユーザーにとって関心のないデバッグ情報であり、アクションは不要です。
- ***Information*** - 無効なプロパティ設定を使用するなどのチャートの不適切な使用が発生しましたが、チャートのパフォーマンスに影響はありません。通常、ユーザーのアクションは不要です。
- ***Warning*** - 破棄されたオブジェクトを使用するなどのチャートの誤った使用法がいくつか発生し、パフォーマンスの低下など、チャートに障害を引き起こす可能性があります。ユーザーのアクションが必要になる場合があります。
- ***RecoverableError*** -回復するはずのチャートでエラーが発生しました。ユーザーは ***ChartMessage*** イベントに従う必要があります。そうしないと、この重大度のメッセージが例外としてスローされます。
- ***UnrecoverableError*** - エラーが発生したため、チャートを回復できませんでした。着信例外を示す場合があります。ユーザーは ***ChartMessage*** イベントに従う必要があります。そうしないと、この重大度のメッセージが例外としてスローされます。
- ***Critical*** - 常に例外として排除されるチャートで重大なエラーが発生しました。

MessageType プロパティはメッセージの基本的な種類を説明し、***Details*** プロパティにはそれに関するより具体的な情報があります。すべてのメッセージタイプは、LightningChart 名前空間に列挙された ***MessageType*** で確認できます。

不要なメッセージは ***chart.Options.ChartMessageMinimumLevel*** プロパティ値を変更することで除外できます。このプロパティでは、設定された最小レベル以上のメッセージのみをイベントシステム経由で送信できます。デフォルトでは、***MessageSeverity.Warning*** に設定されています。

例外は、***ChartException*** オブジェクトとしてスローされます。このオブジェクトには、***ChartMessage*** イベントと同様の例外に関する詳細情報を含む ***ExceptionInfo*** が含まれます。場合によっては、レンダリングエンジンの例外など、グラフが他のタイプの例外をスローする場合があります。重大度レベルが、***MessageSeverity.Warning*** 以上のすべてのメッセージで例外を発生させたい場合、***chart.Options.ThrowChartExceptions*** プロパティを true（デフォルトでは false）に設定する必要があります。

ChartMessage イベントを常にサブスクライブして、チャート内のエラーおよび非公開メッセージからの例外について通知することを推奨します。チャートに問題があり、そのサポートが必要な場合は、アプリケーションに動作中のメッセージ/例外ハンドラーがあることを確認し、***ChartMessages*** をログに記録してサポートリクエストに含めてください。

18. LightningChart® Trader

Trader ライブラリ (Arction.Wpf.TradingCharts.dll / Arction.WinForms.TradingCharts.dll) は、トレーディングおよびファイナンス用アプリケーションを簡単に作成するための管理、ツール、メソッドから編成されています。 トレーディングライブラリは、堅固で高速な LightningChart API を基に構築されています。現在、***TradingChart*** が主な管理をしており、トレーディングアプリケーションを構築するためのプロパティとメソッドのコンパクトなインターフェースを有しています。複雑なエンジニアリングアプリケーションから発生する API オーバーヘッドはありません。現行のバージョンは、WPF および WinForms Forms チャートを含み、UWP トレーディングチャートは、今後利用可能になります。

18.1 TradingChart 作成

TradingChart の使用にあたり、相当するアセンブリをプロジェクトに追加する必要があります。Add Arction.Wpf.TradingCharts.dll (または WinForms 版) をプロジェクトのレファレンスに追加すると TradingChart -objects の作成が可能になります。

```
using Arction.CustomControls.Trader.Wpf;
```

```
// Creating a TradingChart component
TradingChart _chart = new TradingChart();

// Adding chart into the parent container, in this case a grid.
(Content as Grid).Children.Add(_chart);
```

ツールボックスからドラッグして、TradingChart コンポーネントを作成する事も可能です。

18.2 TradingChart 展開

ソフトウェアが展開されるコンピューターで TradingChart アプリケーションを実行するために、展開キーをコードで申請する必要があります。これを通常の LightningChart (第 4.4 章参照)と同じように行います。インターナルチャーティングコンポーネントへのレファレンスを、プロジェクト(*Arction.Wpf.Chart.LightningChart* or *Arction.WinForms.Chart.LightningChart*)へ追加する必要があります。

18.3 インターナル LightningChart コントールの使用

TradingChart は、通常の LightningChart コントロール上で作成されます。 ***GetInternalChart()*** -メソッドで、TradingChart のインターナル LightningChart コントロールとそのプロパティ全てへの直接アクセスが可能になります。従って、両方のチャートの機能を組み合わせる事が可能です。インターナルチャートへアクセスするためには、それぞれのアセンブリへのレファレンスを、プロジェクト(Arction.Wpf.Charting.LightningChart)に含める必要があります。

18.4 UI コンポーネント

シンボルまたは会社名を基に、検索バーでプロバイダー(AlphaVantage.co)からのトレーディングデータの検索が可能です。***ShowSearchBar*** プロパティを無効にして隠す事ができます。

ツールメニューは、チャートの上側右にあります。このメニューは、利用できる全ての作図ツールを含み、チャートのカラーテーマの変更も可能にします。

チャートの時間の範囲は、チャートの下側右隅のボタンを介して変更する事ができます。***ShowTimeRangeSelection*** プロパティは、可視性を管理します。

Volume や ***RelativeStrengthIndex*** のようないつくかのテクニカルインディケーターは、チャートの下の個別のセグメントに描かれます。これらの場合、セグメントスプリッターと呼ばれる水平線は、セグメント間に自動的に描かれます。このラインをマウスでドラッグすると、セグメントの高さが変更できます。

18.5 トレーディングデータの追加

データプロバイダー

Trading Chart はプロバイダーを内蔵しており、シンボルまたはセキュリティ名を基にセキュリティが探せる検索バーがあります。テキストボックスにサーチストリングを打ち込んだ後に、「検索」ボタンを押すか、「Enter」キーを押してセキュリティ検索ができます。その

後、検索結果が検索バーの下側のリストに表示されます。結果をクリックすると、プロバイダーからの該当するトレーディングデータセットを取り込み、それをチャートへ追加します。

データの取り込みは、検索バーを使用しないやり方でも可能です。***OpenSymbol()*** メソッドをコードに使い、グラフの ***Symbol*** プロパティに従ったデータが入手できます。***OpenSymbol()*** は、入手したデータを自動的にグラフに追加します。さらにそれは、通貨などの ***Symbol*** を基にした情報を入手し、それをグラフのタイトルに表示します。

TradingChart は特に設定されていなければ、定義前のデータプロバイダーを使用します。しかし、テストや目的の学習だけの用途には、これらのデータプロバイダーを使用する事を推奨します。その理由は、これらのプロバイダーはすべてのユーダーによって使用され、しばしば、許可されるデータ要求数に制限があるからです。従って、独自の ApiKey、あるいは、最終商品用のデータプロバイダーを使う事を推奨します。定義前のデータプロバイダーを使用するべきでなければ、***DataProvider*** プロパティを ***UserDefined*** に設定する必要があります。

ユーザーが定義前のデータプロバイダーをいくつか使用したいが、独自の API キーを使用したい場合は、***SetRestApiKey()*** メソッドを使う必要があります。これは、トレーディングデータを要求している際に、TradingChart に与えられたキーを使用させます。

ファイルから

TradingChart は、OHLC データの配列を戻す ***GetOhlcDataFromFile()***メソッドを介して、csv ファイルから直接トレーディングデータを読み込む事ができます。 ***GetOhlcDataFromFile()*** メソッドは、次に挙げる列の順でファイルのデータを想定します DateTime、 Open、 Close,、 High、 Low, Volume (オプション)、 OpenInterest (オプション)。 DateTime フィールドは、時間、分、秒を含める必要はありません。 ***SetData()*** メソッドにより、取り込んだトレーディングデータをグラフへ加える事が可能になります。ファイルから自動的にデータを取り込む事で、日付セットの始めと終わりの DateTime 値を基に、グラフの時間の範囲が調整されます。

ファイルからのデータを読み込んで OhlcData 配列を満たす必要はありません。データを取得する場所と方法を決定し、必要なロジックを実行するのは、すべてユーザー次第です。従って、データが OHLC 形式で表現される限り、***SetData()*** を使って外部サーバーからのデータを読み込む事もできます。

18.6 テクニカル分析指標

TradingChart には、複数のテクニカルインディケーターが組み込まれており、取り込んだトレーディングデータを基に自動的に計算されます。テクニカルインディケーターをグラフへ追加するためには、まずは、インディケーターを作成し設定します。次に、それを ***Indicators*** リストへ追加します。このリストは、全てのテクニカルインディケーターを保存するために使用されます。

```
RelativeStrengthIndex rsi = new RelativeStrengthIndex()
{
   PeriodCount = 14,
   HighColor = Colors.Lime,
};
_chart.Indicators.Add(rsi);
```

テクニカルインディケーターを削除するには、TradingChart's Indicator リストからそれらを削除する事で可能です。

```
_chart.Indicators.Remove(indicator);
```

TradingChart には、下記のテクニカルインディケーターがすでに組み込まれています。

- Relative Strength Index (RSI)
- Money Flow Index (MFI)
- Stochastic Oscillator (SO)
- Moving Average Convergence Divergence (MACD)
- Volume
- Open Interest (OI)
- Simple Moving Average (SMA)
- Exponential Moving Average (EMA)
- Weighted Moving Average (WMA)
- Bollinger Band

18.7 作図ツール

作図ツールは、TradingChart 上で自由に呼び出せるビジュアルツールです。すべての作図ツールは、***FreehandAnnotation*** を除き、2 つのコントロールポイントを起点とします。始めのポイントは、作図ツールを呼び出した時に設定されます。2 つめのポイントは、作図を終了した時にマウスの左クリックで設定されます。これらのコントロールポイントは、設定後にマウスで移動する事もできます。作図ツールは、描画の際および、コントロールポイントを移動している時にリアルタイムで呼び出され更新されます。

作図ツールは、built-in Tool メニューを介して、あるいは、***StartDrawing()*** メソッドを呼び出す事でも追加できます。そのメソッドを呼び出した後に、グラフを左クリックするとすぐに作図が始まります。その時に始めのコントロールポイントが設定されます。再度グラフを左クリックすると作図が停止します。

```
// Start drawing a yellow Trend line.
TrendLine trendLine = new TrendLine();
trendLine.LineColor = Colors.Yellow;
_chart.DrawingTools.Add(trendLine);
```

```
trendLine.StartDrawing();
```

マウスでコントロールポイントの 1 つをハイライトし Delete を押せば、作図ツールは削除できます。同様に、インディケーターに対してコードで作図ツールが削除できます。

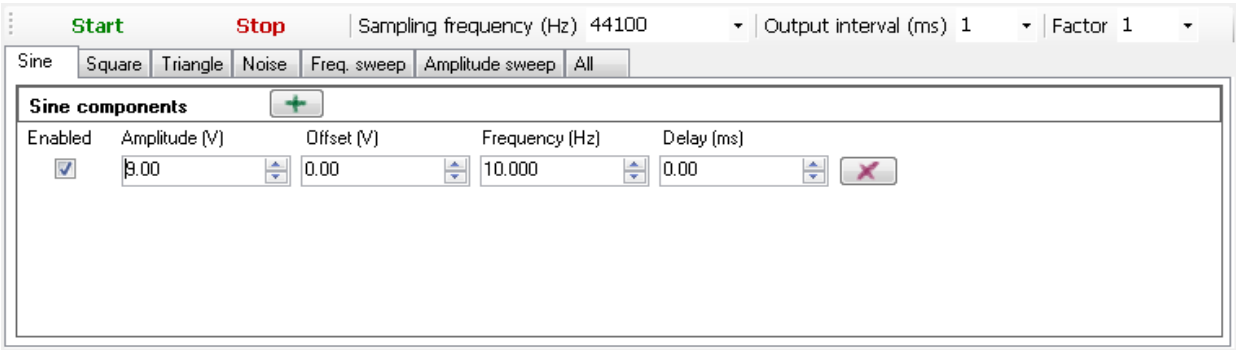
TradingChart には、下記の作図ツールが内蔵されています。

- Trend Line
- Fibonacci Fan
- Fibonacci Retracements
- Fibonacci Arc
- Linear Regression
- Freehand Annotation

19. SignalTools

SignalGenerator

SignalGenerator コンポーネントを使用して、リアルタイム信号を生成できます。信号は様々な波形の合計として生成されます。複数の ***SignalGenerator*** コンポーネントをマスター/スレーブ関係でリンクして、同期されたマルチチャンネル出力を生成できます。***SignalGenerator*** は、LightningChart で信号監視またはデータ収集ソフトウェアを開発するときに非常に便利です。



利用可能な波形は、下記のカテゴリーに分類されます。

- Sine 波形
- Square 波形
- Triangle 波形
- Noise 波形 -ランダムに生成される信号
- Frequency sweeps - frequency 1 から frequency 2 への sine sweep
- Amplitude sweeps - amplitude 1 から amplitude 2 への sine sweep

SignalReader コンポーネント

SignalReader コンポーネントを使用すると、信号ソースファイルからデータを読み取り、選択したレートで再生できます。***SignalReader*** 出力データストリーム形式は ***SignalGenerator*** に似ています。ファイル名を指定して ***OpenFile(…)*** メソッドを呼び出します。ファイル名には、サポートされている形式の拡張子が必要です(wav と sid) 。次に、***Start()***メソッドを呼び出します。

AudioInput コンポーネント

AudioInput コンポーネントにより、ユーザーは Windows の録音デバイスから System.Double 値に信号をキャプチャできます。これらの値は LightningChart でレンダリングされ、AudioOutput コンポーネントに送信されてファイルなどに保存されます。

ソースコードで手動で、あるいは Visual Studio ツールボックスから、フォーム、ウィンドウ、ユーザーコントロールなどにドラッグアンドドロップして、新しい ***AudiioInput*** インスタンスを作成します。***LicenseKey*** プロパティを設定し、***AudioInput*** インスタンスが Windows のレジストリから探すのではなく特定のライセンスキーを使用できるようにすることを推奨します。試用版を使用する場合、***LicenseKey*** プロパティはデフォルト値のままにすることができます。

AudioOutput コンポーネント

AudioOutput コンポーネントにより、ユーザーは System.Double 信号データをオーディオストリームに変換し、スピーカーで再生したりサウンドデバイスのライン出力コネクターに送信したりできます。

SpectrumCalculator コンポーネント

SpectrumCalculator コンポーネントにより時間領域と周波数領域間の変換が可能になります。下記のパブリックメソッドが利用可能です。

- ***CalculateForward***（*double[] samples*、*out double[] fftData*） - FFT を使用して、時間領域の信号データを周波数領域に変換します。出力 fftData にも負の値が含まれています。入力データ配列と出力データ配列は同じ長さでなければなりません。長さはデータの解像度であり、0 Hz からサンプリング周波数/ 2 に広がり、出力値間の周波数間隔は同じです。
- ***CalculateForward***（*float[] samples*、*out float[] fftData*） – 上記の方法と似ていますが、単精度の浮動小数点値用です。
- ***CalculateBackward***（*double[] fftData*、*out double[] samples*） - 周波数領域データを時間領域に変換します。 FFT データから信号サンプルを作成します。サンプル数は、入力 fftData の長さに等しくなります。
- ***CalculateBackward***（*float[] fftData*、*out float[] samples*） – 上記の方法と似ていますが、単精度の浮動小数点値用です。
- ***PowerSpectrum***（*double[] samples*、*out double[] fftData*） - 信号データのパワースペクトルを計算します。 ***CalculateForward*** と同じですが、出力値は絶対値です。

- ***PowerSpectrum***（*float[] samples*、*out float[] fftData*）– 上記の方法と似ていますが、単精度の浮動小数点値用です。
- ***PowerSpectrumOverlapped***（*double[] samples*、*int fftWindowLength*、*double overlapPercent*、*out double[] fftData*、*out int processedSampleCount*）- ソース信号サンプルデータ内の計算ウィンドウをオーバーラップパーセントでシフトすることにより、パワースペクトルを計算します。信号データは、指定された FFT ウィンドウの長さよりも長くなければなりません。出力 FFT データは fftWindowLength の長さであり、ソースデータの長さと必ずしも同じではありません。出力データには絶対値があります。

20. ヘッドレスモード

ヘッドレスモードは、グラフィカルユーザーインターフェイス（GUI）にアクセスせずにデバイスで作業するソフトウェア機能です。「ヘッドレス」という用語は、ソフトウェアが周辺機器（ディスプレイ、キーボード、マウスなど）の存在またはそれらへのアクセスを必要としない場合にも使用されます。周辺機器がなくても、初期化または実行プロセスが失敗することはありません。ただし、この場合ソフトウェアは入力を受信し、ネットワークやシリアルポートなどの他の通信インターフェイスを介して出力を提供する場合があります。

LightningChart SDK には、サービス、コンソールアプリケーション、WPF のクライアントアプリケーションを含むサンプル Visual Studio ソリューション (DemoService.sln) が付属しています。DemoService.sln は、***C:\ProgramData\Arction\LightningChart .NET SDK v.10\DemoService*** フォルダーにあります。

デフォルトで、Windows サービスは、システムユーザーアカウントのセキュリティコンテキストで実行されます。**試用および開発ライセンスのインストールは不可能です。**このため、サービスアプリケーションには有効な**展開キー**が含まれているか、アクティブライセンス（試用版/開発版）を持つ通常のユーザーの資格情報で実行されている**必要があります**。

20.1.1 ヘッドレスレンダリング

ヘッドレス構成では、ヘッドレス/サーバー環境で LightningChart を実行できます。予想されるシナリオには、ユーザーインターフェイス（UI）を使用しないソフトウェアアプリケーションでのバックグラウンドレンダリングと、チャートコンテンツからのビットマップイメージの生成が含まれます。その後、画像をさらにレンダリングするためにヘッドフルシステムに渡すことができます。

ヘッドレスモードを有効にするには、***HeadlessMode*** フラグを ***true*** に設定します。プロパティには、***chart.ChartRenderOptions***（WPF の場合）または ***chart.RenderOptions***（WinForms の場合）

からアクセスできます。LightningChart は Windows サービスタイプのアプリケーションでその使用を自動的に検出するため、モードを指定する必要はありません。

UI と視覚的な parent が欠落している LightningChart の初期化されたインスタンスは、レンダリング要求やレンダリングエンジンを初期化する信号を受け取りません。したがって、ユーザーは、下記の操作と構成をチャートに適用する必要があります。

- ***chart.Width*** および ***chart.Height*** プロパティを使用してサイズを定義する。
- ***chart.InitializeRenderingDevice(true)***を呼び出してレンダリングエンジンの初期化を要求する（WPF のみ）。
- 画像をエクスポートするロジックを実装するための ***chart.AfterRendering*** イベントをサブスクライブする。

チャートはプロパティの変更に引き続き反応します。新しいフレームのレンダリングは、必要に応じて、連続した ***BeginUpdate()*** および ***EndUpdate()*** 呼び出しで照会できます。

レンダリングされたフレームは、次の様々な方法でエクスポートできます。***OutputStream*** プロパティ、***SaveToStream*** メソッド、***CopyToClipboard*** メソッド、***CaptureToByArray*** メソッド、***SaveToFile*** メソッド。 一般に、ビットマップストリームが優先されます。また、ViewXY チャートは ***SaveToStream*** および ***SaveToFile*** メソッドでヘッドレスモードの EMF、WMF、SVG をサポートします。

21. クレジット

Intel Math Kernel ライブラリ

LightningChart®.NET SDK は、高速フーリエ変換メソッドなど、一部の部分で Intel Math Kernel Library を使用します。Arction アセンブリには、このライブラリから作成されたネイティブ DLL ファイルが含まれています。Arction Ltd.は Intel Math Kernel Library を使用するライセンスを取得しています。

オープンソースプロジェクト

以下のオープンソースプロジェクトとマテリアルプロバイダーに感謝します。

.NET 用 DirectX ライブラリ
LightningChart は、Arction 製の拡張機能を備えた SharpDX から派生した DirectX .NET DLL を使用します。http://www.sharpdx.org/

マップ出典
LightningChart®.NET マップは、以下のマッププロバイダーからインポートされています。

世界、北米、ヨーロッパ： Natural Earth、 http://www.naturalearthdata.com/
オーストラリア： オーストラリア統計局、 http://www.abs.gov.au/
アメリカの道路： アメリカ国立地図、 http://www.nationalatlas.gov

Scalable Vector Graphics 出力

LightningChart SVG エクスポートは、RiskCare Ltd による SvgNet プロジェクトコードを部分的に 使用しています。

多項式回帰

多項式回帰計算コードは、Math.Net ライブラリに部分的に基づいています。
http://www.mathdotnet.com/

変更されたソースコードパーツは、リクエストにより Arction サポートから無料で入手できます。（support@arction.com）
オープンソースプロジェクトの著作権表示については、LightningChart SDK インストールフォルダーの ***LightningChart .NET Readme.txt*** を参照してください。